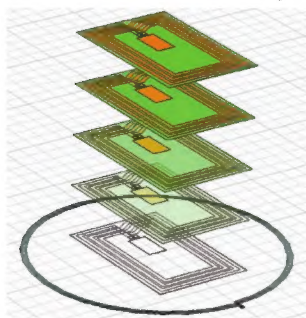
Interface

2004
12
Contents



[表紙デザイン：㈱プランニング・ロケッツ]





特集
ユビキタス社会のコア技術
RFID——無線ICタグのデバイスからシステムまで
Cover Story RFID—From the devices to the systems of the wireless IC tag 83

第1章 規格の確立，法律制度，電波行政の迅速な対応が望まれる
RFIDをとりまく現状 84
Chapter 1 Present situations surrounding RFID
坂下 仁(Hitoshi Sakashita)

第2章 新旧の二つの技術で相互補完するシステム
RFタグとバーコードの併用による自動認識 93
Chapter 2 Automatic recognition using RF tag and bar code
村上 貴一(Takakazu Murakami)

Appendix USB対応リーダ/ライタを使用した
Visual Basicによるアプリケーション開発 99
Appendix Application development with Visual Basic
吉崎 辰美(Tatsumi Yoshizaki)

第3章 アンテナとタグの間の通信を視覚化して把握する
回路・システム・3D電磁界シミュレータによるRFID設計 103
Chapter 3 RFID design with circuit, system and 3D magnetic field simulator
門田 和博(Kazuhiro Kadota)

第4章 T-Engineフォーラムのユビキタスを使った
RFIDのセキュリティとプライバシ 110
Chapter 4 Security and privacy of RFID
尾立 英基/小林 真輔(Hideki Odate/Shinsuke Kobayashi)

第5章 小型基板とインターネットで実現する
H8SマイコンによるRFIDシステムの作成 119
Chapter 5 Making of an RFID system using H8S microcomputer
松永 隆文(Takafumi Matsunaga)

第6章 電波を発するタグによる病院・介護ホーム支援システムの実例
アクティブRFIDによる所在管理システムの実際 129
Chapter 6 Realities of a location management system using active RFID
馬場 功(Isao Baba)

Interface

2004 12

Contents

話題のテクノロジ解説

バーコードよりも大容量，高密度で誤り訂正機能を持つ
二次元コード「QRコード」の概要 138
Summary of a 2D code "QR code"
末武 陽一 (Youichi Suetake)

リンカを100%使いこなそう！(第1回)
リンカとオブジェクト 146
Linkers and objects
坂井 弘亮 (Akihiro Sakai)

組み込み向けストレージ・デバイスとして注目される
フラッシュ・メモリの基礎と電源障害に強いファイル・システムの構築 159
Basics of the flash memory and construction of a power supply trouble tolerant file system
長澤 恒也 (Tsuneya Nagasawa)

処理のあらましを理解して回路を作ろう
わかる！ CRC計算回路の作り方とアレンジ 183
Easy to grasp! How to make and arrange a CRC circuit
森岡 澄夫 (Sumio Morioka)

アナログ回路もプログラマブルな時代へ
プログラマブル・アナログIC"FPAA"のセンサ処理への応用 194
Application of a programmable analog IC"FPAA"for sensor operation
Steve Harrold

モータ制御にART-Linuxを活用した
Linuxによるロボット・アームのリアルタイム制御 198
The realtime control of robot arm using Linux
吉川 智康 (Tomoyasu Yoshikawa)

ショウレポート&コラム

自動認識技術の総合展示会
第6回 自動認識総合展 13
AUTO-ID EXPO 2004
北村 俊之 (Toshiyuki Kitamura)

ハッカーの常識的見聞録
デスクトップにもIntel版64ビット拡張機能 17
Intel 64bit expanded functions for desktop
広畑 由紀夫 (Yukio Hirohata)

IPパケットの隙間から
IMAPに関するトラブルについて 19
A trouble according to IMAP
祐安 重夫 (Shigeo Sukeyasu)

シニアエンジニアの技術草子(四拾五之段)
賢い買い物 232
A smart buy
旭 征佑 (Shousuke Asahi)

一般解説&連載

TOPPERSで学ぶRTOS技術(第10回)
ターゲットへのTOPPERSの移植 168
Porting TOPPERS to the target
邑中 雅樹 (Masaki Muranaka)

プログラミングの要(第18回)
辞書圧縮――フレーズ単位の効率的な圧縮法 174
Dictionary compression――An efficient, phrase-unit compression
宮坂 電人 (Dento Miyasaka)

組み込みプログラミング・ノウハウ入門(第20回)
文章からのクラス抽出――日本語で書かれた仕様書をUMLへ変換する 206
Class sampling from documents ―― converting specification sheet written in Japanese to UML
藤倉 俊幸 (Toshiyuki Fujikura)

「VxWorks」を使ったRTOS技術の基礎と応用(第10回)
組み込みシステムのデバッグ(中編)――VxWorksシミュレータ 215
Debugging of an embedded system(2)――VxWorks simulator
高山 剛 (Takeshi Takayama)

フリーソフトウェア徹底活用講座(第20回)
GCC2.95から追加変更のあったオプションの補足と検証(その8) 220
Supplements to additions and changes in the options from GCC2.95 and their verification(8)
岸 哲夫 (Tetsuo Kishi)

情報のページ

Show & News Digest 15
NEW PRODUCTS 234
海外・国内イベント/セミナー情報 240
読者の広場/読者プレゼント 241
次号予告 242

▶ 自動認識技術の総合展示会

第6回 自動認識総合展

北村 俊之

「HUMAN&AUTO-ID ――生活と産業を支える自動認識技術」をテーマに「第6回 自動認識総合展」が9月15日(水)～17日(金)の3日間，東京ビッグサイトで開催された．主催は(社)日本自動認識システム協会．バーコード，二次元コード，RFID，ICカード，バイオメトリクスなどの分野で，トレーサビリティやユビキタス・コンピューティング，物流や医療支援に必要な自動認識技術や機器，サプライ用品，応用システムまでが一堂に会した展示会となった．例年通り，バイオメトリクスとICカードに関しては「BIOMETRICS EXPO」および「CARD EXPO」として併催された．また，自動認識の基礎から最新動向までを紹介するセミナも同時に開催された．出展数は177社，2団体428小間と，昨年を上回る規模での開催となり，来場者数は，33,957人にのぼった．

● AUTO-ID EXPO

デュプロは，バーコード・シート・リーダ，OCRシステム，カラーラベル・プリンタを中心とした展示を行っていた．バーコード・シート・リーダでは，バーコード伝票の高速自動読取機「SR-1」(写真1)の参考出品を行っており，来場者の関心を集めていた．同製品は，設置場所を問わない小型設計で，130枚/分の読み取りが可能，読み取り済みのバーコード・ラベルに証拠スタンプの印字ができる点などを特徴としている．また，イメージ・スキャナを使用したOCRでカラーとモノクロ帳票の認識が可能なOCRシステム，免許証画像の読み取りに特化したOCRシステムなどの展示も行われた．


写真1 デュプロのSR-1

ゼブラテクノロジーズは，RFIDとモバイル無線プリンティングに関する製品を中心に展示を行っていた．RFID関連では，13.56MHzのISO RFIDのラベルをサポートする，デスクトップ・クラスの「R2844-Z」およびUHFをサポートする「R4MPlus」の参考出品を行っていた(写真2)．「R2844-Z」は，使用現場でスマート・ラベルをコード化できるコンパクトRFID印刷システムで，多目的に使える製品であるという．リスト・バンドやスマート・ラベルなどのスマート・メディアの印刷が行え，標準でシリアル，パラレル，USBの各ポートを持ち，オプションでEthernetの搭載をサポートしている．

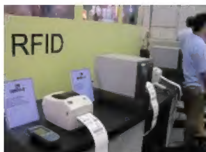
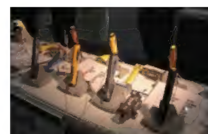


写真2 ゼブラテクノロジーズのR2844-Z(左)，R4MPlus(右)

セイコーインスツルメンツは，モバイル・プリンタ「WhiteTiger(MPU-L465)」(写真3)および「DPU-3445」のデモを行っていた．「WhiteTiger」は，300dpiの解像度を持ち，イメージ・データや高密度バーコード，二次元コードを高精細に表現可能なプリンタである．インターフェースは，Bluetooth，USBおよびシリアルをサポートしており，今後は防水性に優れたレイン・モデルやラベルを自動剥離可能なピーラ・モデルの発売も予定しているとのことだった．


写真3 セイコーインスツルメンツのWhiteTiger(MPU-L465)

ディジ・テックは，コンパクト・ハンディ・ターミナル「Sparklet」シリーズを中心とした展示を行っていた．「DHT-101」(写真4)は，レーザ・スキャナ搭載のペン型ハンディ・ターミナルである．非接触でバーコードの読み取りが可能，単3電池1本で約80時間動作，約80gと小型で軽量なことから人気の製品だという．また，「DHT-200」シリーズとLAN通信BOX「DCC-S15T」を利用して，ハンディ・ターミナルで読み取ったデータをLAN経由で直接サーバに送信するというデモも行われていた．POS端末やパソコンを増設することなく，データの一括管理が行えるソリューションとして注目されているとのことだった．


写真4 ディジ・テックのDHT-101

マーステクノサイエンス/マースエンジニアリングは，釣銭カード対応決済ユニット「KS-05」(写真5)の展示デモを行っていた．カードのリード/ライト時には，認証処理を行うことでデータの漏洩を防止，カードのビジブル部分にはカード残金，ポイント，最終利用日の表示が可能などの特徴を持っているという．紙幣は堅牢なカセット金庫に収納されるため，一切現金に触れることなく回収が可能となっている．また，ActiveXコントロールの提供により，既存のPC-POS向けのアプリケーション開発も可能としているとのことである．


写真5 マーステクノサイエンス/マースエンジニアリングのKS-05

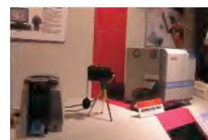
● BIOMETRICS EXPO/CARD EXPO

サイレックス・テクノロジーは，指紋認証ログオン・ソフトウェア「SX-Biometrics Suite」(写真6)のデモを行っていた．同製品は，Windowsにログオンする際のユーザ名やパスワード入力を，指紋認証により行うことができる．利用環境に合わせて，PCカード・タイプ，ICカード・リーダ/ライタ付き，USBインターフェース対応の3タイプの指紋センサが用意されている．本人拒否率は0.1%，他人誤認率は0.001%，照合速度は1s以内などを特徴としており，ICカード併用版(「SX-Biometrics Suite for COMBO」)を利用することで，ICカード内の指紋データとの照合によりユーザを特定する，ICカードを抜くと自動的にスクリーン・セーバ・ロックがかかるなど，より強固なセキュリティを実現できる．最大登録ユーザ数は「SX-Biometrics Suite」で32，「SX-Biometrics Suite for COMBO」で300となっている．



写真6 サイレックス・テクノロジーのSX-Biometrics Suite

アイアンドディは，顔写真入りIDカード発行システム「ID Maker」(写真7)の展示を行っていた．磁気カード，バーコード，Mifare，Felica，TN2などのICカードに対応しており，カード発行プリンタも再転写，ダイレクト，再転写/ダイレクト共用など，運用に合わせたラインナップを揃えている．撮影BOXは，IDカード発行に必要な写真取込みを無人セルフ・モードで行える機能をサポート．また，高価なICカードの再利用を可能にしたシート「ID Sheet」も用意している．カード紛失時に，迅速に再発行を行えることから，大学などからも注目されているとのことだった．


写真7 アイアンドディのID Maker

■最新のショーレポートがWebでも公開されています！
URL： http://www.willcomnet.jp/showreport/

Show & News Digest

Xilinx Virtex-4 ジャパンプレスサミット

■ 日時：2004年9月10日(金)
■ 場所：LEVEL XXI アーバンネット大手町ビル(東京都千代田区)

ザイリンクス社は，Virtex-4 FPGAの正式発表と新組織体制発足の記者発表を行った．新組織としては，DSP市場への本格参入をめざしたDSP事業部と，プロセッサ・コアおよびIP部門を統合したプロセッサ事業部を発足させた．

ワールドワイドマーケティング副社長のサンディ－プ・ビジ氏は「ザイリンクスはPLDの分野ではすでに大きなシェアをもっている．現在はDSPや組み込み向けCPU，ASIC/ASSPが使われているような分野があるが，今後はそのような特定のチップが使われていた分野もPLDへの置き換えを進め，ザイリンクスの市場を広げていく」と語った．

また，製品としてはVirtex-4 FPGAファミリにおいて，LX/SX/FXプラットホームの3種類を展開していく．LXはロジック集積度の向上を主眼とし，SXはDSP性能の向上，FXはハード・プロセッサ(PowerPC)を内蔵した製品となる．LXプラットホームは同日より一般出荷開始，SXは2004Q4，FXは2005Q1に出荷開始予定である．また，開発ツールISE 6.3iも同日より出荷開始された．

Virtex-4は90nmプロセスを採用した製品．同社では90nmプロセスはSpartan-3に続いて二つ目となる．


Virtex-4搭載評価基板

UNL，新型ユビキタス・コミュニケータを発表

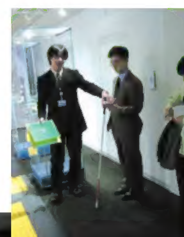
■ 日時：2004年9月15日(水)
■ 場所：YRPユビキタスネットワーキング研究所(東京都品川区)

YRPユビキタスネットワーキング研究所(UNL)は，新型のユビキタス・コミュニケータ(UC)を開発し，実証実験用に生産を開始する．

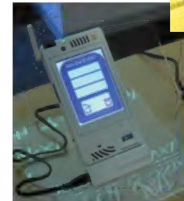
UCは標準T-Engineをベースとし，RFタグ・リーダ/ライタや動画再生，VoIP，指紋認証などのさまざまな機能を内蔵した小型端末である．各周波数に対応したマルチバンドucodeタグ・リーダを採用しているため，さまざまな規格のRFタグを読むことができる．このとき，サーバとの通信にはeTRONをVPNとして使うとのことである．

また，同研究所は国土交通省と協力し，神戸市においてUCを利用したプレ実証実験を9月30日から開始する．これは神戸市内の各所にRFタグを埋め込み，UCを持ち歩くことにより道路や名所，危険物などの案内を行うというもの．UCのもつ音声再生機能により，危険物に近付くと音声が発せられるデモも行われた．

同時に，腕時計型のμユビキタス・コミュニケータも参考出品された．これはμT-Engineベースの腕時計型端末で，同研究所所長の坂村 健氏は，ネクタイの裏側に付けられたRFタグをμUCで読み取ってみせるなどのデモを行っていた．


▲UCを使った交通案内のイメージ．点字板の裏にRFタグが埋め込まれている


▲μユビキタス・コミュニケータ


▲ユビキタス・コミュニケータ

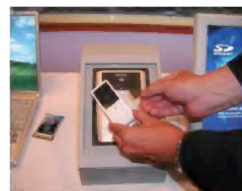
松下電器，ICカード機能を搭載したSDメモリ・カード「smartSD」を開発，12月よりサンプル出荷

■ 日時：2004年10月1日
■ 場所：都道府県会館(東京都千代田区)

松下電器産業(株)は，SDメモリ・カードに，非接触型通信やセキュリティといったICカードの機能をもたせた「smartSD」を開発した．2004年12月よりサンプル出荷を開始し，2005年秋に商品化を予定している．

ICカード部の不揮発性メモリには，EEPROMよりも高速な同社開発の「FeRAM」を用いており，より高速なデータの書き込みが可能となっている．

カード・サイズがSDメモリ・カードと同じものと，miniSDカードと同じものの2種類を用意しているる．前者は，アンテナを内蔵しているが，通信距離を伸ばすためには別途電源が必要となる．後者はアンテナを内蔵していない．


smartSDカードの使用例

英国Wolfson Microelectronics plc新製品発表会

■ 日時：2004年9月13日(月)
■ 場所：LEVEL XXI アーバンネット大手町ビル(東京都千代田区)

英国エジンバラに本社を置くミックスド・シグナル半導体企業Wolfson Microelectronics plcがCODEC ICを含む3製品を発表した．

WM8974は1チップのディジタル・スチル・カメラ用CODEC IC．A-D/D-Aコンバータを内蔵し，音声入出力の両系統をサポートしているほか，自社開発の独自DSPコアによりカメラ本体から発生するモータのノイズをカットするプログラマブル・ノッチ・フィルタなどを実現している．

そのほかの新製品は多機能プリンタおよびスキャナ向けの高速パイプラインA-Dコンバータであるディジタイザ・デバイスWM8216，2Vrmsの高出力ドライブ機能をもつD-AコンバータWM8521．



Wolfson WM8974

ハッカーの常識的見聞録

広畑 由紀夫



今月の常識

デスクトップにも Intel 版 64 ビット拡張機能

☆Intel Xeon シリーズが Nocoma コアとなり，Intel 版 64 ビット拡張機能「EM64T」が実装されました．そこで今月は，Pentium4 版の EM64T を追ってみます．

この夏，Intel 版 64ビット拡張機能（以下 EM64T）搭載の Xeon プロセッサの発売後，デスクトップ向け Pentium4についても EM64T 搭載のプロセッサの登場にかかわる情報が少しずつ流れてきました．Intel 社においても Pentium4の紹介ページに EM64T のものが登場し，今後の製品出荷をにおわせているようです．

● EM64T の動作

EM64T の動作は現在の IA 32と互換性を持たせるための 32ビット・モード（レガシ・モード）と 64ビット・モードが存在し，64ビット・モードにおける 32ビット・アプリケーション動作モード（互換モード）を含めておもに 3種類の動作モードが存在します．

① レガシ・モードでは，現在の OS などはそのまま動作しますが，EM64T の機能を生かすことができません．しかし，従来の CPU からの移行段階で，まずプロセッサをアップグレードして動作させることが可能なため，当面はこのモードが使用される機会が多いでしょう．

② 64ビット・モードでは，現在未発売の Windows XP 64ビット Edition など，64ビット OS が必要になります．また，アプリケーションやデバイス・ドライバも 64ビット専用のものが必要になりますが，EM64T の機能を生かすことができるようになります．現状では，この環境向けのデバイス・ドライバやアプリケーションなどの開発を β 版を通じて先行開発のために使うことのほうが多いでしょう．

③ 互換モードは，64ビット OS 上で 32ビット・アプリケーションを動作させるものです．この場合，OS やデバイス・ドライバは 64ビットのものが使用されますが，アプリケーションは 32ビットのままのものを動作させます．そのために，32ビットと 64ビット API 間のパラメータ互換性などにおいて一部支障がでる可能性もあります．互換モードは，64ビット OSおよびデバイス・ドライバなどの環境が整った後，アプリケーションの 64ビット移行までの暫定的なもの，もしくは資産の有効利用としてとらえるべきでしょう．

● 32 ビット環境からの移行

さて，実際に 32ビット環境からの移行について検討するために，次のスペックで実際に 32ビット版 OSインストール環境に，EM64T と差し替えてみました．

● 使用したおもな環境

マザーボード： ASUS P5AD2 Premium（Intel925X）
HDD： MAXTOR 社製 SATA2− ATA 150対応ドライブ
GFX： Radeon X800XT − 256搭載 PCI-Express 16X
RAM： DDR II-533 512M バイト× 2
OS： Windows XP Pro SP2
Windows XP Pro x64 Edition（64ビット拡張）
PreRelease Build 1218

CPU を載せ換えて第 1回目の起動後，HID 関連のデバイスの再検索が始まり，デバイス・ドライバのインストールが行われている最中にハングアップしました．その後，2度目の起動ではもう一度 HID 検索の後，正常に終了しました．この現象は初期インストールからの試験の際，3回中 3回とも発生したため，デバイス更新される部分があるようです．EM64T への載せ換え作業が終わり，起動が安定してからはアプリケーションなどの動作確認を行いますが，32ビット OS 上なのでアプリケーションは「動いて当然」です．

● 64 ビット環境

さて，気になる 64ビット OS 環境ですが，製品版が存在しないため，プレリリースではありますが，Windows XP Pro x64版（Build1218）のインストールを行ってみました．

プレリリースのため，デバイス・ドライバなどは最低限しかなく，ネットワーク・デバイス用のドライバも入手できなかったため，インストールの確認しかできませんでしたが，先に示した環境でプレリリース版のインストールおよび実行は完了しました．

● 筆者が感じた現在の問題点と今後

今回のテスト中，すでにインストール済みの 32ビット版へのアップグレードに関しては，第 1回起動時のデバイス認識でハングアップという点を除けば大きな問題はありませんでした．しかし，EM64T で初期インストールを行った 32ビット OS 環境では，そのパフォーマンスが生かされないという現象が複数のマザーボードで確認できました．

レガシ・モードにおける「ファイナルファンタジー公式ベンチマーク 2」を例に挙げてみると，先の環境にてスコアが 6500前後は出していたのですが，EM64T にて初期インストールしたものではスコアが 5000を切るなど，大幅なパフォーマンスのダウンが見られました．この現象における原因の特定には至りませんでしたが，ほかのマザーボード環境でも再現されることから，特定可能な原因があると推測されます．

ひろはた・ゆきお　OpenLab.

IPパケットの隙間から

IMAPに関するトラブルについて

祐安 重夫

Mさんから電話がかかってきた．いつもはたいてい電子メールが来るのだが，どうしたのかと思ったらメールが送信も受信もできなくなっているようなので，チェックしてほしいという用件である．たしかにそれでは，メールを出すわけにはいかない．

最近では電子メールが発達したおかげで，電話やFAXがかなり減少している．いや，正確に言うとかかってくる電話やFAXのほとんどがセールスの電話や宣伝や広告のFAXになっている．とくにFAXなど，こちらからどこかに送ることもなくなりつつあるので，ほとんど無用の長物となっている．

さて，本題に戻ろう．こういう時代だから，電子メールが使用できなくなるというのは，けっこう深刻な問題である．日本でJUNETがスタートした時代から電子メールを使用している筆者などは，インターネット・メールがそれほど強固な信頼性を持ったものとは思っていないし，当時は電子メールの商用利用が制限されていたのだが，現在では電子メールは，電話やFAXよりずっと重要な商用インフラストラクチャとなってしまっている．

Mさんのところのメール・サーバにログインして，MTUのログをチェックしてみたところ，たしかにここ数時間，メールの受信も送信もできていない．ためしに`df`コマンドを実行してみたところ，驚いたことに`/var`以下を100%使いきっている．これではメール・スプールもメール・キューもいっぱいなので，送信も受信もできないのは当然である．原因の追求より，とりあえず空き領域を確保しなければならない．`/var`以下にはfmlのスプールもあったので，メーリング・リストの中から大きなメールの交換が多く，保存しておく必要のないアーカイブをいくつかまとめて削除した．それで，キューにたまっていたメールは配送されたし，メールの受信も行われるようになってきた．

とりえず回復したので，次は原因の追求である．ためしにメール・スプールの`ls`をとってみたら，驚いてしまった．半数以上のユーザのスプールが10Mバイト以上あって，なかには30Mバイト以上などというケースもある．これで犯人は判明した．IMAPである．

そもそもこのメール・サーバを最初にインストールしたときは，IMAPはまだ実験的なプロトコルだったし，IMAPに対応したMUAも限られていた．POPなら，サーバから読み出せばスプールは空になる．そのため最初にパーティションを切ったときに，デフォルトでも`/var`にはそれほどの領域は確保されなかった．fmlの使用なども考慮して多めに確保したつもりだったのだが，IMAPのここまでの影響は予測の範囲外だった．

とりあえず，すでにいなくなっているユーザがいないかを確認して，そのスプールを削除するなどして，もう少し領域を確保した．そのうえで，不要なメールはサーバに残さずに，こまめに削除してもらうようにお願いした．

IMAPの利点の一つは，会社のサーバに保存してあるメールを外からも参照できるという点である．これで出向先や家からも，仕事がしやすくなる．もっとも欠点がないわけではなく，この件の数日後にMさんのところの女性社員から電話がかかってきて，会社でメールをいくら読んでも，しばらくすると同じメールが未読になってしまうといわれたときには，少し頭をかかえてしまった．

どうやら話を聞いてみると，家のPCの電源を切らずにうっかり出社してしまったらしい．家のPCは定期的に会社のメール・サーバに接続して，その特定のメールを未読にリセットしていたようだ．こういうちょっとした問題は起こるが，基本的にはIMAPは有用な技術である．

さて，こうしてしばらくは安定していたメール・サーバだが，また同様の問題が起こった．今度の原因を追求していくと，どうやら今度は，社内メールで巨大な画像ファイルが流通しているのが原因らしい．社内でならファイル・サーバが使用できるので，そこに画像ファイルを置いてそれをメールで知らせればすむのだが，家で仕事をしていて作成した画像ファイルは，メールで送信するしかないということのようだ．

ここで問題なのは，送信者がそれを自分自身にもコピーをCCで送信していたことだ．現状の設定ではsendmailのデフォルトのメール・キューを使用しているので，CCすると単純計算でもメール・キューとメール・スプールで`/var`以下に，元の画像ファイルの3倍の領域を一時的に使用することになり，当然のごとくパンクしてしまう．

とりあえず画像を送信する際は，自分自身にCCしないことをお願いして，これもなんとかしのいだ．`sendmail.cf`の設定でメール・キューは別の場所に移すことができるが，それをするには一度メール・キューが空になったことを確認して，サーバをLANから切り離してから行わないと危険である．そこまでやるなら，いっそパーティションを切り直して，`/var`そのものを大きくするのが有効な対策ということになるだろう．

もう少ししたら，その方向で検討をはじめざるをえないだろう．

すけやす・しげお　インターメディア・アクセス

昨年以来注目されているRFID——無線ICタグに関して，末端のデバイスからRFIDを用いたシステムの実際までを解説する．

さまざまなところで取り上げられる機会が増えたRFIDだが，その規格および実装系は多くの団体・企業が関わり，さまざまなものが存在する．電波の周波数，デバイスの形状，デバイスの特性…．これらに関して，第1章では，規格の動向に関して全体を俯瞰した解説を行う．

第2章以降はRFID技術の各論に移る．電磁界シミュレータを用いたRFID機器の設計，各種RFIDリーダ/ライタを用いた開発例などを解説する．RFIDというと，とかく大規模なシステムが想起され，手を出しにくいというイメージがあるが，現在ではUSB接続によるRFIDリーダ/ライタも入手でき，開発もVisual Basicで行えるなど，敷居は低くなっている．本特集を参考に，実際にRFID開発を試してみていただきたい．

さらにRFIDでは，セキュリティやプライバシの問題も懸念される．これらの問題の多くは経路上での通信の傍受によるものに起因するが，経路を強固に暗号化することにより，問題の発生を防ぐことができる．その実例についても紹介する．

最後に，RFIDは近未来の技術ではなく，すでに「食品トレーサビリティ」や「病院・介護ホーム支援システム」などのシステムが現場で実験/導入されつつある．そのような事例についても解説を行う．

特集 ユビキタス社会のコア技術

RFID

——無線ICタグのデバイスからシステムまで

1 規格の確立，法律制度，電波行政の迅速な対応が望まれる
RFIDをとりまく現状 坂下 仁

2 新旧の二つの技術で相互補完するシステム
RFタグとバーコードの併用による自動認識 村上 貴一

Appendix USB対応リーダ/ライタを使用した
Visual Basicによるアプリケーション開発 吉崎 辰美

3 アンテナとタグの間の通信を視覚化して把握する
回路・システム・3D電磁界シミュレータによるRFID設計 門田 和博

4 T-Engineフォーラムのユビキタス IDを使った
RFIDのセキュリティとプライバシ 尾立 英基/小林 真輔

5 小型基板とインターネットで実現する
H8SマイコンによるRFIDシステムの作成 松永 隆文

6 電波を発するタグによる病院・介護ホーム支援システムの実例
アクティブRFIDによる所在管理システムの実際 馬場 功



規格の確立，法律制度，電波行政の迅速な対応が望まれる

RFIDをとりまく現状

坂下 仁

RFIDは現在までに多くの種類の規格が提案され，実用化され，流通している．それらはさまざまな形状とさまざまな周波数，さまざまな通信方法を用いているなど，複雑な面が多い．また，RFIDを用いたタグ（RFタグ）は電波を用いることから電波法などの各種法規制に準拠する必要があり，RFタグが付いたままの状態で輸出入を行う場合には，各国間の調整も必要となる．本章では，これらRFIDの現状と法規制などについて解説する．

また，RFIDはプライバシの問題が懸念され，RFIDの採用に反対する団体も存在するなど，普及にあたっては考慮しなければならない点も多い．そこでプライバシの問題に関しても言及する．（編集部）

最近，電子タグ，無線タグ，ICタグ，あるいはIDタグといったさまざまな用語が，各種団体や新聞紙上などで用いられています．これらは「RFタグ」の機能や，類似技術の関連であることを意識しての表現だと思いますが，いずれも同一物に対する異なる表現であり，混乱のもとになっている観があります．

また，この業界ではこうしたことばの錯綜だけでなく，無線の周波数による特性の違い，各国の法的環境（電波法）の違い，タグ製造元やリーダ/ライタ製造元などの思惑の相違や得手不得手，あるいはタグの製造方法の相違などが複雑に絡んでいます．さらに，RFIDに対する社会の認識にもばらつきが大きく，一部では実態と大きくかけ離れた議論もなされているようです．

このことは，RFIDに対する将来への希望と，それに対応して将来出てくるべき技術論とが十分に議論されていないことが原因になっているといえます．

こうした現状に対して，いま必要なことは，一つは規格の確立であり，もう一つは法律制度，電波行政の迅速な対応です．本章ではこのあたりについて，説明を加えたいと思います．

実は冒頭に述べた用語に関しては，（社）日本自動認識システム協会が日本規格協会から受託し，委員会を組織してJIS化したものがあります．RFID関連の用語は「JIS X 0500 データキャリア用語」に記載されており，ここではRFIDシステムにおけるタグを「RFタグ」と定義しています．「RFID」は，いわば“技術”を表す用語であり，部材・機器としてRFタグとリーダ/ライタ

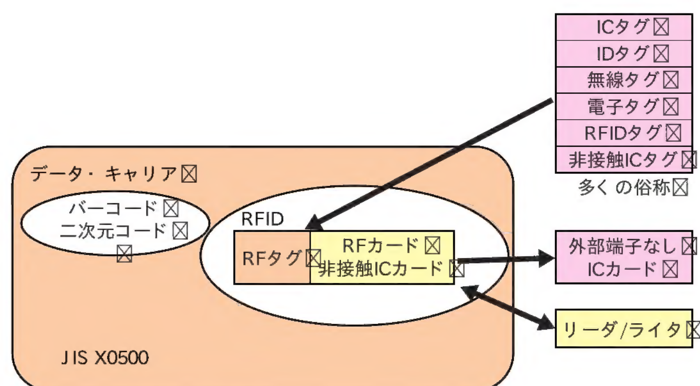


図1　RFID関連用語の現状

表1　JIS X 0500データキャリア用語

番号	用　語	定　義	対応規格（参考）
10001	データキャリア	情報を人，動物，または物に付加し，人，動物，または物を特定するために利用する情報担体の総称．RFID，一次元シンボル（バーコード），二次元シンボル（二次元コード）などを表す	Data Carrier
10023	RFID	誘導電磁界または電波によって，非接触で半導体メモリのデータを読み出し，書き込みのために近距離通信を行うものの総称	RFID（Radio Frequency Identification）
10024	RFタグ	半導体メモリを内蔵して，誘導電磁界または電波によって書き込まれたデータを保持し，非接触で読み書きできる情報媒体	RF Tag
10025	能動型RFタグ	みずからデータを送信する機能を備えているRFタグ	Active RF tag
10026	受動型RFタグ	リーダ/ライタから送られてきた搬送波の電力を利用して送信する機能を備えているRFタグ	Passive RF tag
10027	リーダ/ライタ	RFタグのデータを書き込み，読み出しする装置．通常，アンテナと制御装置で構成する	Reader/Introgator
10028	アンテナ	リーダ/ライタの一部で，RFタグとの物理的に電磁界ないしは電波の送受信を行う誘導素子放射部分（空間結合素子部分）	Antenna
10029	交信	RFタグとリーダ/ライタ（アンテナ）間の無線通信	Radio communication

があります．

図1にRFID関連の用語の現状を，表1にJIS X 0500データキャリア用語を示します．

本章の構成を，以下に示します．

1) RFタグをめぐる最近の動向
2) 各種団体の動向
3) 国内における電波行政の動向（とくにプライバシ問題，電波法，電波利用税について）
4) 人体への安全性（とくにペースメーカなどの医用機器への影響調査，人体への防護指針について）
5) ISOでの検討状況（とくに欧米を中心にSCM関連で進んでいるEPCグローバル検討状況について）

さて，RFID技術を用いたシステムの開発，RFタグの提案がますます推し進められ，ユビキタス時代に向けて，RFタグによる生活の利便性の向上がおおいに期待されています．しかし一方で，RFタグの現在の実力と，あるべきとされる姿とのギャップは依然大きいといえます．

ユビキタス時代に使用されるRFタグのモデルは非常に単純化されたものとして描かれていますが，実は考慮しなければならない前提の要因として，周波数や出力などの電波要因，リーダ/ライタの形状や寸法，RFタグ（別称：電子タグ，無線タグ，ICタグ）の形状や寸法，構造（電池の有無など）といった多くのことが挙げられます．

1 RFタグをめぐる最近の動向

● 使用できる周波数は決められている

国内電波法の関係で，日本で使用できる電波の周波数帯は135kHz，13.56MHz，2.45GHzが主流となっています．世界に目を向けると，日本ではまだ使用されていないUHF帯の電波を使うタグの使用が検討されています．日本でも同じ動きは進みつつあり，その優れた特性を活用できる用途での使用が期待されています．

図2から，日本で比較的多く使われている13.56MHzがタグのサイズなどの面で優れた性能をもっていること，長距離用としてはUHF帯に期待がもたれることがわかります．ただし，法規制によっては通信距離や読み取り性能が大きく変わってくるので，2005年春に向けて検討されている電波法の審議を待って，米国，EUと同レベルの性能が発揮できるのかを見極めたうえで，国内における性能に期待したいところです．

RFタグはSCM（サプライ・チェーン・マネジメント）[注1]の主流になることが想定されています．ウォルマート，DoDなどの2005年における動きがどのように進展するかがおおいに注目されます．なぜなら，このようなSCM導入の流れは，一国内レベルにとどまらず，国境を越えて物品が移動することを想定する必要が生じてくるためです．

同時に，日本のRFタグをとりまく環境も，これらの世界の動きに追随する必要が出てきます．

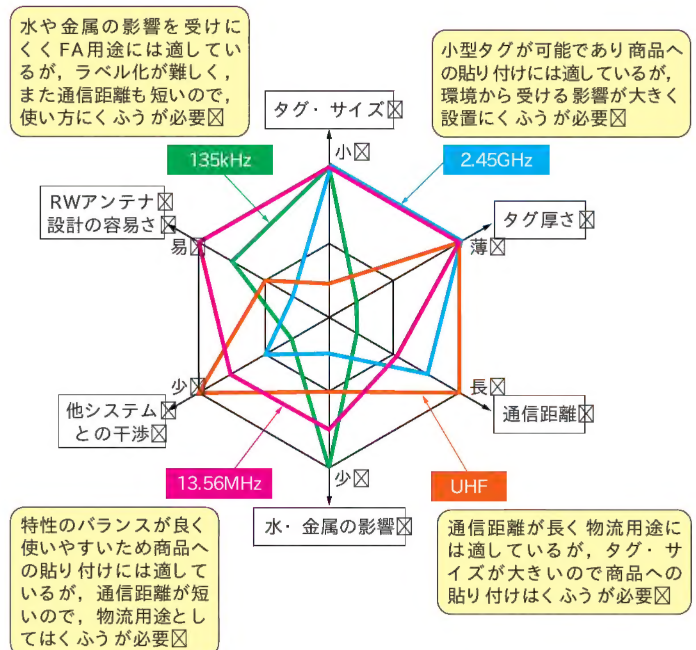


図2 RFタグの周波数帯別の特徴比較

▶日本国内で商品や容器に貼付されたRFタグは，海外のどこででも読み取り可能

▶海外で貼付されたRFタグは，日本国内の電波環境でも読み取り可能

以上のようなことを実現するためには，海外の電波法と国内の電波法との間に大きなずれがなく，同様の性能を発揮できる環境が確保される必要があります．

現在，総務省の指導で委員会が開催されており，2005年に向けて，基本的な技術を日本国内でも活用できるための動きとなっていくことを期待したいところです．

● アクティブ・タグとパッシブ・タグの使い分け

RFタグには，アクティブ（電池内蔵のタグ）とパッシブ（電池なしのタグ）の二つのタイプがあります．RFタグの寿命やコストを考えるとパッシブが望ましいといえますが，電波からのエネルギで動作するパッシブに比較して，アクティブは内蔵電池から送信エネルギを得るので通信距離を長くすることができます．

現在の国内電波法では，パッシブで実現する通信距離はせいぜい1m弱程度であり，アクティブでは10m近くとなります．電波法改定の審議によっては，アクティブなら数百mの範囲で通信できるシステム構築も可能です．一般的にアクティブの場

注1：SCMとは，在庫の削減と経営効率の向上を実現するための経営管理手法の一つで，原材料メーカから最終製品メーカ，そして卸売，小売までを一つの連鎖ととらえ，コンピュータを使って総合的に管理することで全体の最適化を図っていくもの．

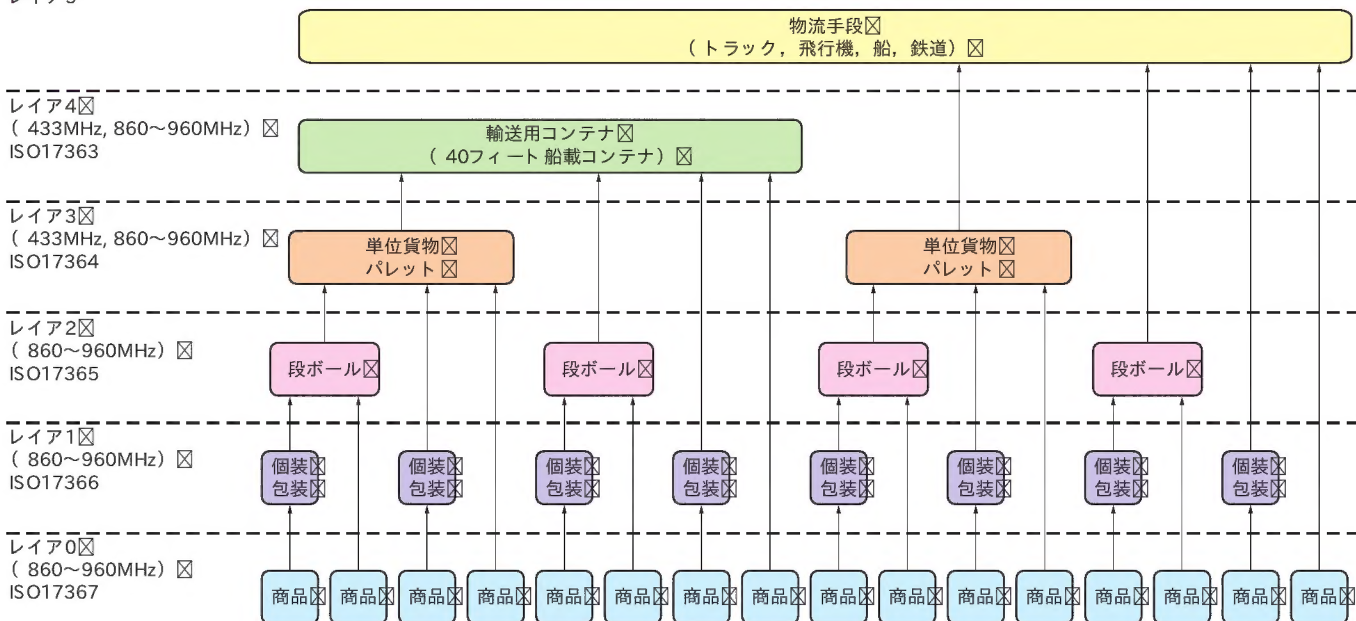


図3　物流管理の階層と RF タグ



図4　EPC グローバルの Web サイト
（http://www.epcglobalinc.org/）

合，電池のサイズが比較的大きく，その分 RF タグが大きくなりますが，セキュリティ用途などの限定的な使い方ではすでに利用が進んでいます．

● 物流管理における RF タグの階層分け

図3に示すように，商品，個装包装，段ボール，パレット，輸送用コンテナといったような，物流における階層によって，使用される周波数やタグの内蔵電池の有無は異なってきます．

比較的下の階層においては通信距離は短めでよく，RF タグが個々の商品に貼付できることが採用において重要なポイントとなります．上のほうの階層になると，たとえば輸送コンテナをヤードの中で位置管理する必要が生じるため，前述したアクティブ型 RF タグを用いた RTLS（Real Time Locating System）など長距離通信が可能なシステムが求められます．

物流管理においては階層レベルによって RFID に対するニーズや考え方は変わってくるため，ISO 規格の中ではタグの種類を階層別に変えていくことを想定しています．

2 各種団体の動向

RFID にかかわる動きを整理すると，大きく三つに分類できます．

一つは ISO を中心とした流れであり，バーコードや二次元シンボル（二次元コード）との関連で進められているものです．十分に時間をかけて検討がなされており，現実的かつ実際的な規格化をめざしています．

二つ目は MIT オートアイディーセンターの流れを汲むもので，現在は EPC グローバル（図4）に一体化されています．「5セント・タグ」といった非常にセンセーショナルな話題を提供し，安価なタグを前提とした運用を提案していますが，実現するまでにはまだ道のりが遠いようで，RF タグに関するスペックや提案しているコストの現実化などがはっきりと見えていません．

今年の初旬，五つのグループでしのぎを削っていたものが，この10月中には一つにまとまるようです．各グループの良いところを取りまとめた形になれば，今後の期待を集めるものとなりますが，多数の会社のいわば妥協の産物，あるいはそれぞれの会社の損得が絡むとすれば，性能，コスト，運用面において当初提案したようなメリットを出せるとは考えにくいというの

New Products ── 産総研，IPv6 対応の 1CD Linux「KNOPPIX/IPv6」を開発，国際認証 IPv6 Ready Logo を取得
（独）産業技術総合研究所は，IPv6 に対応した Linux「KNOPPIX/IPv6」を開発し，国際認証 IPv6 Ready Logo を取得した．これは KNOPPIX 日本語版に（株）アルファシステムズが WIDE プロジェクトで開発した IPv6 プロトコル・スタックを導入したもの．

が実情です．製品ができあがり，今後の提案がどのようなものになるかをしっかりと見極める必要があります．

そしてもう一つの流れが，日本のT-Engineの動きです．ここはRFIDの主流というよりはTRONを中心にしたアプリケーションに，より重きを置いたグループのように思えます．筆者はRFタグでなくても，バーコードや二次元シンボルでも運用できる基本概念であるような気がしています．このあたりはこの後の章で述べられていくので，読者の方のご判断をいただければと思います．

さて，RFIDのリーダ/ライタ製造元にとっても，RFタグ製造元にとっても，最終的には「規格化」がビジネスの大きな鍵となります．他者との互換性をもって使用することができないシステムを，幅広いユーザが採用するとは思えないからです．そこで，メーカ側からISOへの提案が必要となってきます．現にEPCグローバルもISO/IEC 18000-6がすでに確定したというのに，そのAmendment（修正）として新規に提案を開始しました．こうした提案をとおして，その仕様が少しずつ明らかになるはずなので，語られている夢の世界が果たして現実のものとなるのか，その動きに注目したいところです．

RFタグについては上記の三つの動き以外に，各国の思惑も絡んでいます．一つは，米国とEUを中心にした上記ISOの動きであり，大きな資本と長い歴史に裏打ちされています．もう一つは，米国のベンチャ企業を中心に提案されているUHF帯をメインとした提案です．米国の，国を挙げてのバックアップもあり，提案内容と製造に関する完成度に多少の難がありますが，今後の動きは期待されます．そして，これらの動きに呼応して，日本での「響プロジェクト」が今年6月に開始されました．2年間の日本の国家プロジェクトが，どの程度の成果を示すのか，またEPCグローバル対応のRFタグとなることをうたっているが，実際にどのような性能となるのかが期待されます．

3 国内における電波行政の動向

● プライバシとセキュリティの問題

ここ1～2年，「個人情報が，RFタグによって漏洩するのではないか」との議論が活発になってきています．これにより，現実にRFタグの利用にブレーキが掛かるような状況も生じました（コラム1を参照）．RFタグが個人情報を筒抜けにしてしまうというCASPIANの主張はセンセーショナルですが，現在の技術ではこれは非常に困難なことであり，杞憂といえます．しかし，この議論を受けてウォルマート，ジレット，メトロなどが軒並みRFタグの実験を中止あるいは縮小し，個人が特定されない用途，すなわちSCMのような製造元と卸，小売の間での物流用のコンテナ，パレットなどにおける実験に変わりつつあります．

この点での日本の行政の動きは早く，対応策も道理にかなった対応となっています．詳細については「電子タグに関するプライバシー保護ガイドライン」が平成16年6月8日に総務省より発表されています．これは技術的にあり得るかあり得ないかといった議論ではなく，原則的にどうすべきかの議論となっており，RFタグの技術に詳しくない運用業者，商品販売業者，個人などに対して対応方法をわかりやすく説明したものとなっています．そのポイントをいくつか挙げると，

- 電子タグが装着されていることの表示など
- 電子タグの読み取りに関する消費者の最終的な選択権の留保
- 電子タグの社会的利益などに関する情報提供
- 電子計算機に保存された個人情報データベースなどと電子タグの情報を連係して用いる場合における取り扱い

などとなっています．

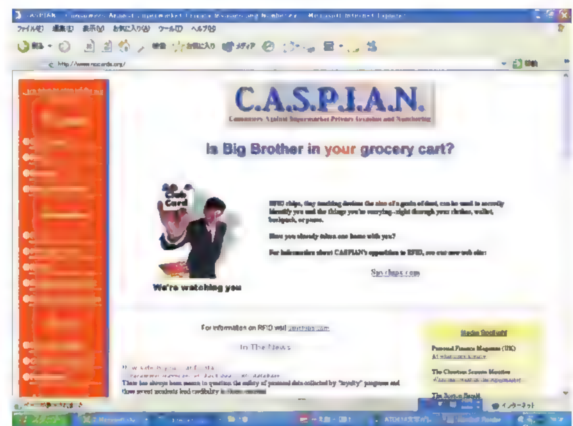
● 電波法改正の動き

RFタグの電波として，これまで日本国内でおもに使用されてきたのは，先述のとおり135kHz，13.56MHz，2.45GHzの3種類です．国際標準（ISO）もこの周波数帯を中心に進められてきましたが，米国からUHF帯の電波を利用したRFタグの提案が行われ，交信距離が10m程度に達するなど，その性能が従来のRFタグに比べてかなり優位性があり，一気に注目を

COLUMN 1

CASPIAN――ICカードに含まれている個人情報の流出，操作に反対するNGO

CASPIAN（カスピアン：Consumers Against Supermarket Privacy Invasion and Numbering）とは，おもにICカードに含まれている個人情報の流出，操作に反対するNGOである．その活動の一環として，Webサイトを見るとウォルマート，メトロのフューチャーストアなどでの運用状況を調査し，批判を続けており，それを受けた運用縮小という影響が出ています（図A）．



図A　CASPIANのWebサイト
（http://www.nocards.org/）

浴びることとなりました．ISOでは900MHz帯と433MHzのRFタグのエア・インターフェースがすでに決定しています．

日本では900MHz帯は，環太平洋群であるITU-Rの地域3に入り，ISMバンドとなっていないため，携帯電話が専用で使用している周波数です．当初，国内での利用は難しいと考えられていましたが，総務省が2003年度に開催した「ユビキタスネットワーク時代における電子タグの高度利活用に関する調査研究会」の報告の中で950～956MHzの周波数をRFタグに開放する意向を示したことから，UHF帯のRFタグ利用への期待が一気に高まりました．

2003年度に経済産業省の支援で行われた各業界の実証実験でも，UHF帯のRFタグを用いた各種の実験が行われ，従来の周波数に比べUHF帯の優位性を示す結果が発表されています．ただし，米国の電波開放域とほかの各国のそれに開きが多く，米国以外での利用時の性能評価が急がれます．

また，総務省の平成16年6月30日付の報道資料に，「移動体識別システム（UHF帯電子タグシステム）の技術的条件」の審議開始（情報通信審議会での審議開始）が掲載されています（http://www.soumu.go.jp/s-news/2004/040630_6.html）．

● RFIDにも電波利用税がかかるか

電波利用税の支払いについては従来，放送，携帯電話などの通信事業者が専用電波帯での利用を進めてきました．RFIDはISMバンドと呼ばれる，共用帯域での利活用であったため，対象とされていませんでした．しかし，現在世界中で使用が検討されているUHF帯では，日本は環太平洋群諸国であるITU-Rの地域3にあたり，ISMバンドから外れていたため専用帯域としてすでに携帯電話で使用されており，その帯域を空けることにより，利用税をどのようにするかの議論が始まりました．

総務省では，一昨年1月から「電波有効利用政策研究会」に「電波利用料部会」を設置し，電波利用料制度について見直しに向けた検討が進められ，意見の募集がなされてきました．行政側の見解としては，公共性の有無（防災無線，放送など），電波の利用形態（専用/共用），使用エリアの大きさ，帯域幅の大きさ，電波の逼迫している場所での使用かどうかなどについての議論を必要としています．さらに使用料の徴収の検討もなされているようです．

しかし業界側からは，

1) ユビキタス社会実現への阻害となること
2) 非関税障壁の懸念があること
3) 型式認定量などと，電波利用税とで二重課金になる心配があること
4) RFIDはベンチャ企業が多くその育成への阻害となること

などを理由に，8月行われたパブリック・コメントに対して，電波利用税に対する反対意見が提出されています．

COLUMN 2

UHF帯のRFタグの利用環境整備

上記の調査研究会で，総務省は2004年度末までに環境整備をする方針を示していました．その方針に基づき，今年度に入ってから総務省は情報通信審議会に検討の推進を示し，小電力無線システム委員会の下部WG（UHF帯電子タグシステム作業班）でRFタグの利用にあたっての具体的な技術基準の検討を開始しました．

（社）日本自動認識システム協会は，RFタグ・システムを市場に提供している各社の代表として，RFID専門委員会を中心にUHFタグのあるべき技術基準について検討を行い，UHF帯電子タグ・システム作業班に委員登録を行い，積極的に主張を行っています．すでに数回のWGが開催され，検討が開始されています．現在，既得権をもっている携帯電話側との主張にはまだ大きな隔たりがありますが，実利用に即し，RFタグが本当に望ましい形で利用できる技術基準の構築に向けて，今後も検討と意見表明を行っていきます．

なお，併せて433MHz帯のRFタグ（バッテリ内蔵型）の検討も同じWGで進められています．総務省は今年度中の環境整備を目指しており，日本国内でのUHF帯RFタグ利用は射程圏内に入ったといえます．



4 人体への安全性

ユビキタス時代にあってRFタグを大量に使用する際に想定される各種の問題について，すでに議論が開始され，行政からの動きやNGOの動きが見られています．各国の対応に比べて，日本の対応はきわめて早いものといえます．

● RFID機器と医用機器との共存

昨年度，ペースメーカなどへの影響の有無を調査研究する場が持たれ，国内のRFID機器の製造元28社が参加して試験が行われました．その結果を受けて，電波干渉によるRFID機器（リーダ/ライタ）から医用機器への過干渉を少なくするために，RFID機器にシールを貼り，ペースメーカ装着者に注意を促すなどの対応を行っています．視認性を考慮して，リーダ/ライタの形態に対応した2種類のシール・デザインを採用しています．

▶ゲート型リーダ/ライタ

ゲート型は，ゲートを通過するRFタグを装着した人や物品を管理します．ペースメーカ装着者などが「RFIDシール」が貼られたゲートを通過するときは，止まらずに通過すれば悪影響が生じません（図5）．

▶ハンディ型リーダ/ライタ

取扱者が手に持って操作するハンディ・タイプのリーダ/ライタについては，ペースメーカ装着者などが一定以上の距離に近づかなければ悪影響が生じません（図6）．

なお，本年度も引き続き，据え置き型とその他機器に関して，同様の試験が行われる予定です．

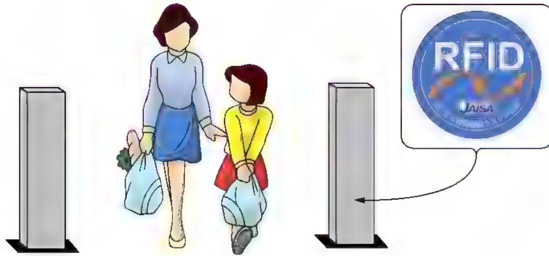


図5　ゲート型への「RFIDシール」



図6　ハンディ型リーダ・ライタへの「RFIDシール」

● 人体防護指針

ARIB STD-38，ARIB TR-T11に詳細が載せられており，10kHzから300GHzという非常に広範囲の周波数帯での電磁界を対象として，RFタグに限定しない電波利用全般に基づいた適用がなされています．海外の規格類との整合性も検討されており，定期的に見直しがなされています．

● 廃棄問題

自動車のリサイクルや家電のリサイクル，リユース，リデュースの3Rといった用途で積極的にRFタグを使用し，廃棄問題に関して積極的に取り込もうとの議論があります．一方，包装材そのものにRFタグを貼付した際，ごみとしての廃棄をどうするかといった議論もあります．

現段階では，これらそれぞれによって対応が異なっています．また，業界や用途によって廃棄物に占めるRFタグの重量比がまったく異なるため，用途ごとのガイドラインが必要となるものと思われます．産業ごとのガイドラインはある程度揃っているので，利活用の状況によって今後，議論すべきものととらえています．

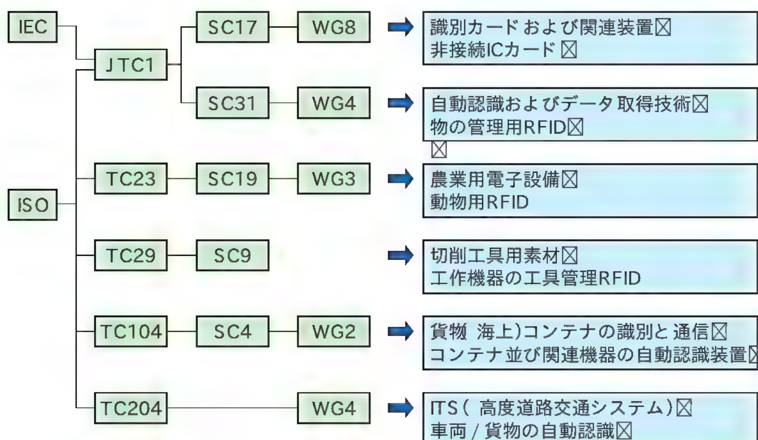
 ISOでの検討状況

自動認識技術の中で，言うまでもなくRFIDは信号伝達処理を電波で行うという特徴をもち，到達距離，メモリへの情報の付与，汚染に対する強さ，遮蔽物の透過などの利点から，今後の使用が期待されています．

データの読み取り/書き込みを，無線でのやり取りで簡便に扱えることはたいへんに便利ですが，逆に電波に特有の問題もあります．ちょうどラジオから流れてくる音楽に雷や違法無線のノイズが入って，せっかくの音楽が台無しになってしまうことがあるように，RFタグの通信においても，ノイズや混信による情報の信頼性がまず問題となります．また，FMラジオとAMラジオに音質の違いがあるように，RFタグにも多くの種類の使用周波数があり，通信特性を生かした用途ごとの使い分けの必要性が生じます．さらに，国によってラジオ，TVの周波数の割り当て方に違いがあるように，RFタグの周波数，変調方式なども国や地域によって異なり，ある国のRFタグが隣の国では法律に触れて使用できないといったことが生じて，国際間の物流に大きな制限をかけてしまう心配もあります．

こういったさまざまな問題を避けるために，国際間のレギュラトリ(規格類)の審議が不可欠といえます．ISOとIECの合同委員会(JTC1)では，SC-31のWG4のなかでRFタグの物流に関する審議を1998年から進めています．本章では取り組まれている規格類の策定に関する状況について報告します．

なお，SC-31以外でのRFIDに関係する検討部門は，図7に



ISO　: International Organization for Standardization 国際標準化機構
IEC　: International Electrotechnical Commition 国際電気標準会議
JTC1 : Joint Technical Committee 合同専門委員会
TC　: Technical Committee 技術委員会
SC　: Sub Committee 分科委員会
WG　: Working Group 作業グループ

図7　RFID関連の検討部門

表2　RFID関連の国際規格の終了状況

ISO委員会	審議中	IS成立
ISO/IEC JTC1/SC17/WG8		ISO/IEC 10536 密着型ICカード
		ISO/IEC 14443 近接型ICカード
		ISO/IEC 15693 近傍型ICカード
		ISO/IEC 10373-6，-7テスト方法
ISO/IEC JTC1/SC31/WG4	ISO/IEC 15961，15962コマンドなど	FDIS 15963（タグ固有ID）
	PDTR 24710（ElementaryTag）	
	ISO/IEC 19789 API	
ISO/IEC JTC1/SC31/WG4/SG3	NP 18000-69 Amendment（EPCglobal仕様）	ISO/IEC 18000-1～7エア・インターフェース（-5を除く）
	FDIS 19762-1，-2，-3（用語）	
ISO/IEC JTC1/SC31/WG4/ARP		TR 18001アプリケーション要求
	TR 24729（ARP 2nd）	
ISO/IEC JTC1/SC31/WG3/SG1	TR 18047-2，-6，-7 RFIDコンフォーマンス	TR 18047-3，-4 RFIDコンフォーマンス
		TR 18046 RFIDパフォーマンス
ISO/IEC JTC1/SC31/WG5	NP＊＊＊＊-1，-2，-3	

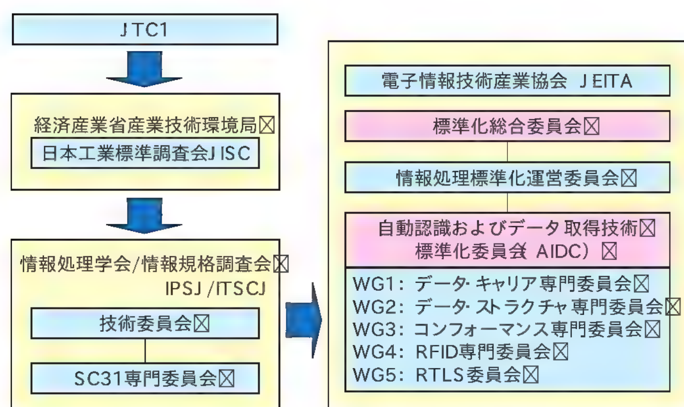


図8　SC31国内審議体制

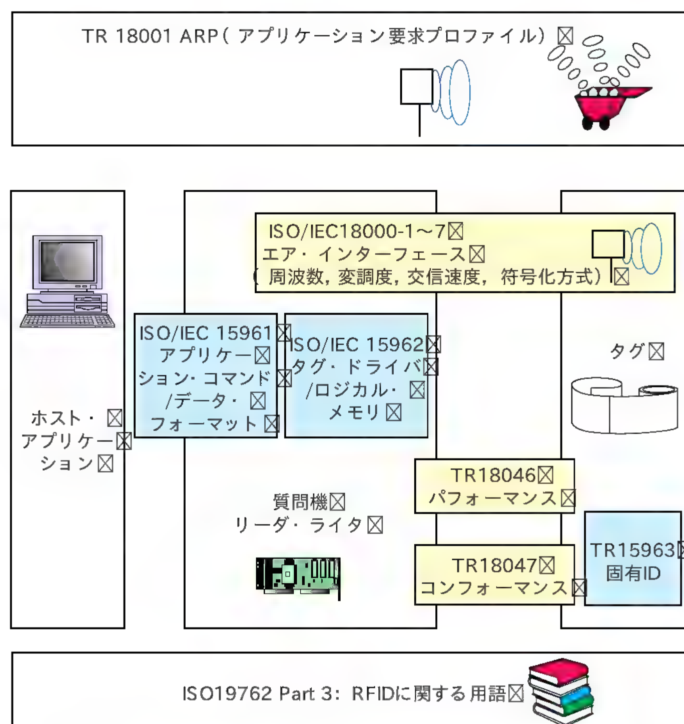


図9　SC31の各規格類の関係

示すとおりです．また，それぞれの規格の終了状況を表2に示します．

● 日本のレギュレトリへの審議体制

国内の審議機関としては，図8に示すように，日本工業標準調査会のなかの情報処理部会にSC31専門委員会があります．そして，実質的な審議は電子情報技術産業協会（JEITA）のなかの自動認識・データ収集技術標準化（ADC）委員会が，いくつかのワーキング・グループに分かれて作業を行っています．

● 各規格類の関係

各検討規格の関連を図9に示します．RFタグの信号をリーダ/ライタが読み取り，それをPCで処理するイメージと併せて，それぞれの検討規格がどの部分にどのように関係しているかを示しています．

▶ TR18001

アプリケーションにおけるRFタグ設計のための要求条件を定めており，具体的なフィールドにおける複数タグの読み込み，タグとリーダとの位置関係あるいは速度と読み取り時間などを定めようとしています．

▶ ISO/IEC 18000

この規格の中では，各種用途におけるエア・インターフェース，すなわちRFタグとリーダ/ライタ間の伝送を定めています．インターフェースは各国の電波法と密接な関係が出てくるところであるため，その点については後述します．

現在審議されている周波数は，135kHz以下（18000-2），13.56MHz（18000-3），400MHz（18000-7），915MHz（18000-6），2.45GHz（18000-4）であり，ISO化が決定しました．なお，5.8GHz（18000-5）はCD投票で規格を検討中止しました．

なぜ周波数の審議がこのように多いのか，簡潔に数種類の周波数で統一されないのか，なぜRFタグが簡潔に数種類の周波数帯で統一されないのかという疑問があるでしょう．これは周波数ごとの特性，水や金属の影響，伝送速度などの電波特有の特徴と，人体防護の観点，医用機器への影響，目的とする用途

News Flash ――電子マネー「Edy」サービスが幕張新都心，札幌地下街，ユニバーサル・スタジオでも開始
電子マネー「Edy」のサービスが，千葉県の幕張新都心や北海道の札幌地下街「オーロラタウン」と「ポールタウン」，大阪の「ユニバーサル・スタジオ・ジャパン」，「なんばグランド花月」などの四つの吉本興業施設でも開始された．

と他用途の混信防止のための周波数の差別化など，さまざまな理由が挙げられます．さらに，微小IC回路の設計，インレットの製造方法，RFタグ・アンテナ設計など数々の要素がRFタグに関与しています．これらに使用者側，タグ製造元，IC設計者の思惑も加わり，当初は非常に多様な種類の提案が行われていましたが，現在はこれでもかなり絞られてきています．さらに各国の電波法が異なるため，各国での使用に耐えるためにはいくつもの周波数を用意しなければなりません．

▶ ISO/IEC 15961，15962

15961の中ではタグとホスト間のアプリケーション・コマンドとアプリケーション・レスポンスとを規定し，バーコードとの整合性を持たせようとしています．15962では質問機のロジカル・メモリとタグ・ドライバを規定しており，ISO/IEC 18000で定めるエア・インターフェースをタグ・ドライバにもたせるコマンドが検討されています．

▶ ISO/IEC 18046，18047

18046ではRFタグを使用する際の機器選定のためのパフォーマンス特性と試験方法がまとめられ，報告書が完成しました．18047ではタグとリーダの位置関係を定義し，モデル化する作業を行っており，一部の作業が終了しています．

RFタグはカードと違い，用途により形状・寸法が異なるうえ，読み取り距離などの運用も違っていますが，SC17でなされているICカードの試験方法と類似の方法を使うことで審議に入っています．

▶ ISO/IEC 15963　固有ID

ISO/IEC 15963の中では，RFタグに64ビットの固有のIDをもたせて，それぞれのタグを区別しています．このIDを使って，たとえばアンチコリジョン処理のなかで各タグの認識手順について定めることができます．

▶ ISO/IEC 19762 Part3 RFIDに関する用語

用語の統一性がなされないと，概念を言い表す際の整合性が取れなくなるため，実はこれは非常に大事な作業といえます．現在も用語の審議が進んでおり，技術の進歩に見合った用語の定義を行っています．しかし，実際には審議が大幅に遅れており，今後の見通しがまだ立っていません．なお，同様の作業は先述のとおり，JIS X 0500: 2002データキャリア用語でも規定されており，国内では表記方法についての最新の技術を適宜見直すことにより，整合性がもたれています．

▶規格の審議状況

規格の審議内容はRFタグの用語，試験方法，RFタグの定義，使用周波数，工業会ごとの利用形態などにわたっています．規格類の審議はかなり進んでおり，2004年中にはほとんどの規格が確定し，まさにRFタグの世界中での大量使用に向けての標準化の基礎が据えられています．

● 各国のレギュレトリ（規格類）の動き

▶ 13.56MHz

日本では13.56MHzの規格の見直しが2002年9月になされ，

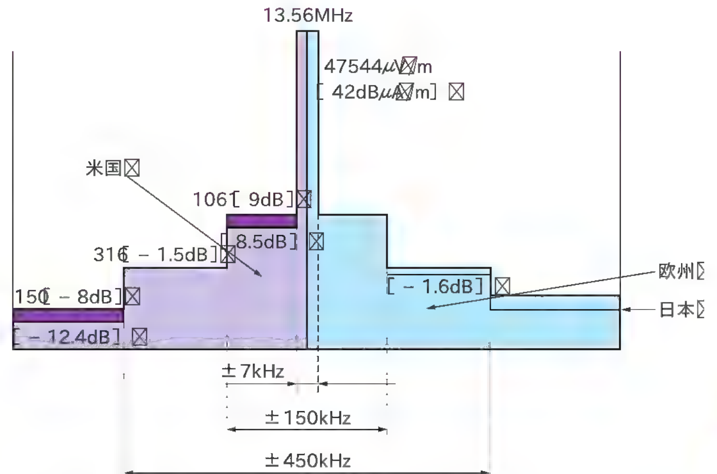


図10　13.56MHzにおける日米欧の比較

図10に示すように欧米と肩を並べる状況になっており，各種用途での展開が可能となりました．しかし欧州ではETSIが13.56MHzの送信レベルを42dBμA/mから60dBμA/m（約1.5倍）へ上げることに関してSE24委員会で検討し，人体への影響を含めて問題のないことを確認，次回SE24委員会で承認される予定となっています．欧州での速い動きは驚くべきものといえます．

▶ UHF

同SE24委員会では，加盟国から提案された出力2WのRFID用UHF帯（865～868MHz）を審議中であり，2003年9月のETSI委員会に提案予定となっています．日本でも2003年6月にUHF帯の検討（950～956MHz）が始まり，早い時期に制度化される期待がもたれています．米国の902～928MHz（FCC 15.247，FCC 15.249）と欧州，日本が現時点では違いがあり，今後の調整が待たれるところです．

ところでここにきて，中華人民共和国，大韓民国，台湾，シンガポール，香港などでのUHF開放が非常に速いテンポで進んでおり，各国における使用可能性が強まり，世界的規模での利用／活用の道が大きく開けつつあります．

▶ 2.45GHz

二つのモードでの審議がSC31で進んでおり，日本では2003年3月に省令改正がなされ，ARIB STD-T81が発行されて，日米欧間ではほとんど制限なく使用できる状況となっています．現状の日本の状況を図11に示します．

▶その他

このほかにもアメリカを中心に提案が開始されているUWB（Ultra Wide Band）については，AdHoc会議が2002年9月にシカゴで開催され，調査継続中です．ITU-Rでも検討が開始された段階であり，まだNPの段階にはほど遠いとはいえ，米国ではすでにFCCの中で規定がなされています．

こうしたRFIDに関するきわめて新しい技術が海外から押し寄

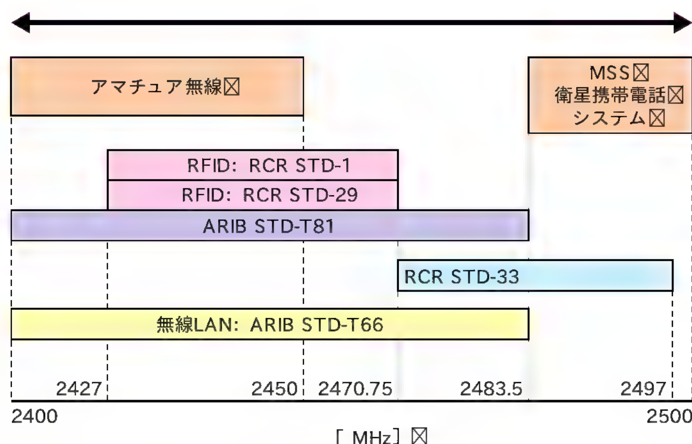


図11　2.45GHzにおける日本の電波法の関係

せ，また規格化の動きは欧米ではきわめて速いものといえます．日本が世界の中で孤立することなく，新しい技術を自由に使えるフィールドを準備する努力が引き続き必要とされています．

おわりに

世界でも日本国内でも，非常に多くの周波数帯を対象に，多くの規格が審議されています．一見むだなことにも見えますが，国際的な利用の中で使用できる選択肢が増えることは，技術革新が進む中で，用途に合った選択が可能となり有益なことです．

日本の電波利用についても，昨年9月の13.56MHzの改正，2.45GHzの整備などが進み，さらにUHF帯の検討が開始されるなど，今後の動きにも期待したいところです．RFIDという新しい技術を使用した産業分野を支える規格類の整備が，引き続き必要とされています．

参考文献・参考URL

(1) 柴田 彰；SC 31(Automatic Identification and Data Capture Techniques/自動認識およびデータ取得技術)総会報告 SC 31専門委員会，社団法人情報処理学会 情報規格調査会 2003年7月
(2) RFタグの開発と応用－無線ICチップの未来－，2003年2月，シーエムシー出版
(3) AIDCセミナー資料，2002年7月，社団法人電子情報技術産業協会自動認識およびデータ収集技術標準化委員会
(4) 森川 和徳；RFID技術の国際標準化と規制 日欧米の規制について，Omron Technics，Vol.42 No.4，2002
(5) JEITA　RFID実践セミナー資料，2004年3月11日
(6) 日本工業出版　RFIDセミナー資料，2004年9月8日
(7) ARIB　Standard　RCR STD-38 2.0版，社団法人　電波産業会
(8) ARIB　Technical Report　TR-T11 1.0版，電波産業会
(9) ISO　http://www.iso.ch/
(10) 総務省　http://www.soumu.go.jp/
(11) 社団法人電波産業会　http://www.arib.or.jp/
(12) 欧州無線事務所　http://www.ero.dk/
(13) 米国FCC　http://www.fcc.gov/
(14) プライバシー保護ガイドライン，平成16年6月8日 http://www.soumu.go.jp/s-news/2004/pdf/040608_4_b.pdf
(15) CASPIAN関連　http://www.spychips.com/
(16) 電波の医用機器等への影響に関する調査結果 http://www.soumu.go.jp/s-news/2004/040618_2.html
(17) 電波の利用状況の調査・公表制度 http://www.soumu.go.jp/s-news/2004/040722_1.html
(18) 移動体識別システム(UHF帯電子タグシステム)の技術的条件の審議開始　http://www.soumu.go.jp/s-news/2004/040630_6.html

さかした・ひとし
リンテック(株)アドバンストマテリアルズ事業部門 情報通信材料部 副部長
(社)日本自動認識システム協会　RFID専門委員会委員長
(社)電子情報技術産業協会　AIDC　SC-31委員会WG5(RTLS)委員長
E-mail：h-sakashita@post.lintec.co.jp

New Products —— KDDI，RFタグ・リーダを搭載した携帯電話を開発，CEATEC JAPANで展示
KDDI(株)は，RFタグ・リーダを搭載した携帯電話を開発し，CEATEC JAPANにて展示した．また，同社の携帯電話用のメタノール燃料電池の展示も行った．



新旧の二つの技術で相互補完するシステム

2 RFタグとバーコードの併用による自動認識

村上 貴一

RFIDは電波を用いてタグとリーダ/ライタ間の通信を行う．これは離れた位置からでも非接触でデータの収集を行えるということを意味するが，これには「人間が動作を目視できない」，「RFタグが破壊された場合，データの復旧が困難」という弱点も存在する．

そこで本章では，従来から用いられているバーコード技術を併用することにより，RFIDの弱点を補完する手法について解説する．また，多数のRFタグの連続発行を行うシステムに関しても紹介する．（編集部）

1 RFタグの採用を検討するにあたっての課題

● 通信距離やコストだけではない

RFタグに対する昨今の急激な認知や関心の高まりと比較して，その普及が期待されたスピードより遅れをとっているといわれています．これは，一般的に海外と比較して厳しいとされる国内電波法の規制下におけるリーダ-タグ間通信距離の不足や，バーコード・システムと比較してコストが高いという点に起因するといわれてきました．

しかしながら，通信距離については運用の最適化によって解決できることが多く，むしろ後述するように，通信距離が伸びることによって問題が発生するという懸念も存在します．また，たびたび主犯とされているコストについては，「単価数円」というタグの実現性以前に，そもそも単価の議論がされるときの「RFタグ」の定義自体が，ICチップ，実装回路，最終形状タグのどれを指すのかが，つねにあいまいであることはすでに指摘されているとおりです．しかし，いずれにせよ，今やRFタグ導入のメリットと導入コストとが絶望的にかけ離れている時代ではなくなりつつあります．

用途の開拓や運用方法の最適化が今後さらに求められるのは当然のことながら，むしろこれまでの実証実験を含む導入現場において決定的に問題とされてきたのは，タグの「発行」と「装着」に絡む部分といえます．

●「発行」と「装着」の重要性

各種RFタグ・システムを実際に運用するにあたっては，おおまかに図1のような工程を経ることになります．

この中でももっとも期待されるのは，当然のことながら④の「運用」がRFタグ特有の各種性能によってどれだけ効率化されるか，またそれがどれだけのコストで実現されるか，ということです．

しかしながら，図1のフローを見るだけでも，④の運用に至るまでに内蔵ICチップへのデータのエンコードなどを行う「発行」と，RFタグを実際に管理対象物と一体化させる「装着」という2工程が必要であり，工数だけをとらえてもこれらが重要な位置を占めていることがわかります．必要とされるタグの枚数が多ければ多いほど，「発行」，「装着」の各工程を効率化しなければ全体の流れが停滞することになり，また，適切に情報入力されたタグが正しく管理対象物に装着されなければ，もっとも重要なシステム自体の精度が，「運用」開始以前に破綻してしまうことを意味します．

さらにコスト面から見ても，管理対象物への装着に際して，ホルダなどの装着器具や接着剤などによる装着作業が必要となれば，とかく問題にされがちなタグ本体の価格に加えて，一層の時間とコストを要することになります．

このような考えに基づき，円滑なRFタグ・システム運用を行うために，「発行」と「装着」という二つのプロセスにおける効率化と利便性向上に着目すると同時に，より信頼性の高い運用環境を提供するため，「破損時リカバリ」が重要になります．

●「発行」と「装着」を同時に実現する手法

いままでのバーコード・システムに必要な粘着加工技術，印字技術，システム化技術などを活用しながら，RFIDシステムにおけるアンテナ設計からICの実装，タグ加工までをトータルにまとめ上げることが要求されるようになってきました．

粘着ラベル・タイプのRFタグと，タグ表面への可変情報印字も可能なRFエンコーダ搭載型のプリンタとの組み合わせ

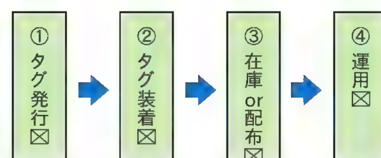
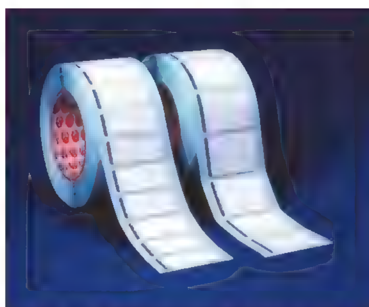


図1 一般的なRFタグ・システムの運用フロー



写真1　粘着ラベル・タイプのRFタグ〔リンテック(株)〕



(a)粘着ラベル・タイプのRFタグ



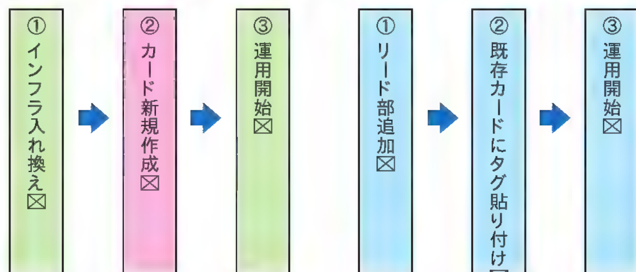
(b)RFエンコーダ搭載型プリンタ

写真2　RFタグとRFエンコーダ搭載プリンタ〔Britem（ブライテム），リンテック(株)〕

(写真1，写真2)で，用途に応じた最適なリーダ/ライタ環境の選定まで，トータルなシステム提案を行えるようになっています．

コラム(pp.96-97)に示すように，システムのテスト導入・評価用パッケージとして，RFタグ×2ロール(各500枚/1ロール)およびタグ発行用プリンタ，リーダ/ライタ，そして専用ソフトウェアで構成される開発キットが発売されています．

幅広い分野での応用が可能で，かつ手軽にスタートできるシステム環境として，「今すぐ使えるRFタグ」を提案できます．



(a)通常のICカードへの移行フロー　(b)粘着タグを使用した移行フロー

図2　既存のシステムからの移行フロー

2 粘着ラベル・タイプの優位性

これまでに示したように，RFタグの供給形態を粘着ラベル・タイプ(ロール形状)に特化したメーカもあります．その理由としては以下の点が挙げられます．

1) RFタグは物理的な形状(外形規格など)を問われない
2) RFタグといえども可視情報の付与は必要である
3) RFタグは連続発行に適した形状・サイズであることが望ましい
4) RFタグは管理対象物への装着適性に優れた形状・サイズであることが望ましい
5) RFタグは使用環境を考慮した形状・サイズであることが望ましい
6) RFタグは破損時リカバリ手段を持つ必要がある

次節以下では，各要求項目との対比において，粘着ラベル・タイプ・タグのメリットを述べます．

● タグの外形について

すでに広く認識されているとおり，RFタグはリーダ/ライタとの間でなされるデータのやり取りを電波によって非接触で行うため，その通信が成立する限り，従来の磁気カードのように定められた外形規格を要求されません．したがって，その外形は実際に使用する際の使いやすさと，通信性能の確保に基準を置くべきであるといえます．

別の見方をすると，必要な通信性能を持つIC実装回路を用途に応じた保護構造をもって管理対象物に装着すれば，装着された対象物自体を，いわば自動認識可能な「タグ」化できるということがいえます．装着するIC実装回路が粘着ラベル・タイプであれば，剥離紙からはがして貼り付けするだけで，さまざまなものを手軽に「タグ」化することが可能となります．

貼り付けするだけでその対象物を「タグ」化できる一例として，既存の磁気カードが挙げられます．磁気カードに粘着ラベル・タイプRFタグを貼り付けすれば，その瞬間から磁気カードはICカードになります．

図2に示すように，通常のフローと比較して，粘着タグを使用したフローの場合，インフラの変更およびカードの差し替えが簡易かつ低コストで実現されます．さらにいえば，名刺などでさえも簡易ICカード化することが可能になります．

● 可視情報の必要性について

RFタグは内蔵ICに対して電波を使用して非接触でデータをやり取りするため，品名やナンバ，バーコードなどの可視情報を付与する必要がないと考えられがちな面があります．しかし，使いやすく精度の高いシステムを構築するためには，以下の理由で何らかの手段による可視情報の付与が必要といえます．

▶ 管理対象物への確実な装着

所定の情報がエンコードされたRFタグを正しく管理対象物へ装着するためには，タグ表面に何らかの可視情報が付与されていることが望ましいといえます．発行者と装着者がつねに同一とは限らず，またかりに10枚のタグがそれぞれ固有の情報を入力され，それぞれが何の外観上の識別がなされる手段をもたないとき，個々を正しく対象物に装着することがいかに困難か

表1　RFタグに要求される形状およびサイズ

機　能	要求仕様	形　状
耐久性	耐衝撃	保護ケース
	耐水	耐水プロテクト
	耐薬品	耐薬品プロテクト
	耐熱	要材料検討
携帯性	間接携帯	カード
		キー・ホルダ
		ストラップ
	直接携帯	リスト・バンド
装着適性	貼り付け	ラベル
	構造的装着	各種ホルダ
再利用	脱着適性	脱着ホルダ
可視情報	印字適性印字スペース	ラベル
		カード
		その他

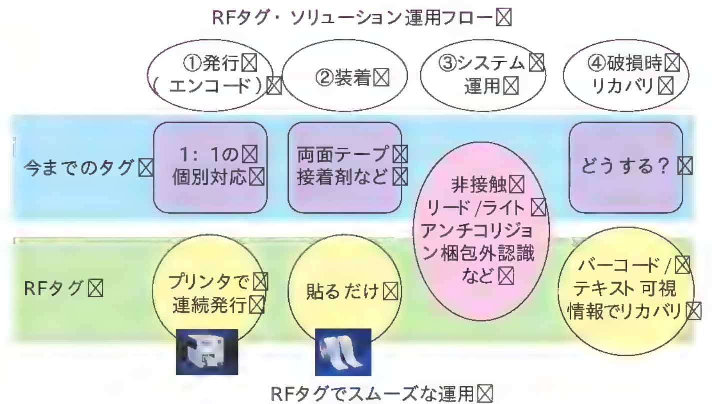


図3　RFタグの発行と装着，破損時リカバリ
スキャニング効率を飛躍的に向上させるRFタグだが，スムーズな運用のためには「発行」と「装着」そして「リカバリ手段」が必要

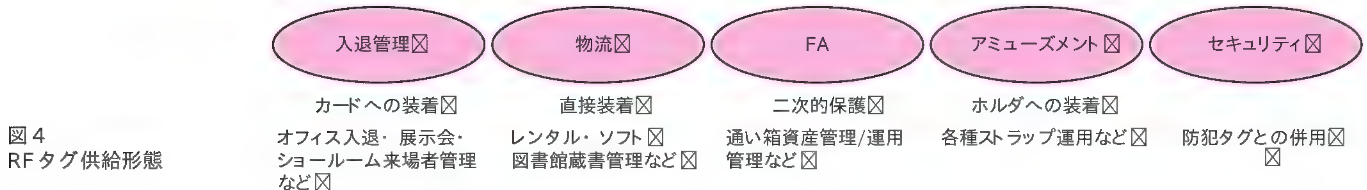


図4
RFタグ供給形態

を想像するだけでも，その必要性は明らかです．

▶ エンコード内容の検証

所定の情報が内蔵ICに正しくエンコードされたか否かを外観で識別するためにも，発行時の可視情報付与は有効な手段といえます．書き込みエラーが発生した際には，外観上識別可能なパターンを付与することにより，エラーの発生したタグを誤って管理対象物に装着してしまうミスを未然に防ぐことが可能です．

▶ バーコード・システムとの混在期解決

バーコード・システムからRFタグ・システムへの移行の際，新規投入アイテムに対してはあらかじめRFタグを装着すればよいのですが，すでに運用されているもの，あるいは在庫品に関しては，バーコード・ラベルからRFタグへの貼り替え，もしくはタグの追加貼り付けが必要となります．その場合，すべての在庫品への装着が完了するまでの一定の期間については，バーコード・ラベルとRFタグが混在することになるため，円滑なシステムの移行のためには，RFタグにも既存システムに対応できる可視情報（可読文字/バーコード）を持たせることが望ましいといえます．

● 連続発行に適した形状とサイズ

すでに述べたように，RFタグが期待されているような幅広い用途で活用されるためには，「発行」，つまり内蔵ICへのデータのエンコードおよび可視情報の付与が確実かつスムーズに行われなければなりません．そのためには，タグの供給形態があらかじめ発行業務の効率性に配慮されたものである必要があります．

タグの発行効率を上げるには「連続発行」が望ましく，粘着ラベル・タイプ（ロール形状）という供給形態は，こうしたことへの配慮もあってのことです．専用のRFエンコーダ搭載型プリンタを用いて，高い発行効率を実現できます．

● 装着適性や使用環境への考慮

RFタグに要求される形状およびサイズについては，必要な性能に応じておおまかに表1に示すような機能があると考えられます．

一見ラベル・タイプの適応範囲は狭いようにも見えますが，価格，ロット，リピート対応などを考えると，カードやストラップ，リスト・バンドといったIC内蔵の専用形状タグで運用することが必ずしも得策とはいえません．先ほど述べたように，粘着ラベル・タイプのタグは，貼り付けするだけで管理対象物や装着器具を容易に「タグ」化できるため，著しく強固な保護構造を必要とする用途を除いて，大部分の用途をカバーできます．この利便性と汎用性こそが，粘着ラベル・タイプのRFタグにとって最大の特徴といえます．

タグ・サイズについては，管理対象物が順次増えていくこと，また装着の際のハンドリング適性などを考えても，最低限必要とされる可視情報の印字エリアが確保されていることを条件に，可能な限り小さいことが望ましいでしょう．そのため，RFタグに適したサイズは20×55mmととらえて製品化されています．

● 破損時のリカバリ手段確保

RFタグが破損した際にどのようなリカバリ手段を確保するかという点も非常に重要なテーマです．

粘着ラベル・タイプRFタグは，薄型であることが要求され

1
2
App
3
4
5
6

るため，保護構造は必要最低限にとどめざるを得ません．しかし一方で，これまで述べてきたように用途および使用環境がそのほか形状のタグと比較して多様であるため，万が一の破損の可能性を考慮しておくことが必須といえます．

そこで，破損時におけるリカバリ手段の確保という観点からも，タグ発行の際にデータのエンコードと同時に行われるバーコードなどの可視情報の付与を推奨しています．

3 ユーザに優しいRFタグとは

● リカバリ・フローの完備と広い適用範囲

これまで述べてきた概念を図3(p.95)に示します．RFタグ採用に当たって検討すべきテーマの多くがここに網羅されており，これらが解決されれば円滑なシステム導入が可能となります．また2項で述べたような適用範囲の広さを表すのが図4(p.95)です．

具体的にシステム導入にあたっては，以下の作業で済みます．

1) 管理対象物に直接貼り付け
2) 貼り付けしてから保護施策
3) 保護構造を施してから使用
4) 使用に適した装着器具に貼り付け

専用形状のIC内蔵タグをそのつど検討するのではなく，上記のような方法を採用すれば，より幅広い用途への展開が可能となり，リピート対応などもスムーズになります．

COLUMN

Britem TS/DC series 開発キット

利便性が高く応用範囲の広い粘着ラベル・タイプRFタグ「TS-L series」＋ロール形状のラベル・サプライを使ってタグの連続発行が可能なRFエンコーダ搭載型プリンタ「DC-P301」という組み合わせで，さまざまな用途でのシステムの提案ができます．

10月1日から「TS-L series」2ロール(各500枚/1ロール)と「DC-P301」1台，(株)エフイーシー製のリーダ/ライタ1式，そして専用のソフトウエアをセットにした，RFタグ・システム開発キットが発売されています(写真A)．

同開発キットは，非接触RFタグおよび関連機器「Britem TS/DC series」のテスト導入およびシステム評価用のオールインワン・システム・パッケージで，低コストで容易にテスト環境の構築が可能です．また，用途に合ったリーダ/ライタ環境の拡張および運用ソフトウェアのカスタマイズにより，本格システムへのスムーズな移行も可能です．

● RFタグ

通信周波数13.56MHz(短波帯)，フィリップスセミコンダクターズ社(オランダ)製ICチップ「I・CODE 1」および「I・CODE SLI」を採用した粘着ラベル・タイプRFタグ「TS-L101」「TS-L102」を各1ロール同梱しています．



写真A　RFタグ・システム開発キット

● タグ発行環境

RFエンコーダ搭載型「DC-P301」およびタグ発行用ソフトウェアは，導入後すぐにRFタグ発行環境を確立することができます．運用環境に合わせた印字レイアウト(バーコード・可読文字)およびICへのエンコード(ICメモリ・マップ割り付け)が可能で，既存バーコード・システムからRFタグ・システムへのスムーズな移行を強力にサポートします．

● システム検証環境

リーダ／ライタ環境((株)エフイーシー製リーダ/ライタ「MRW-3010-Ant」)および検証用ソフトウェアは，ICデータの内容確認はもちろん，POS管理，入出庫管理，出退勤管理などを想定した動作状況の確認が可能です．検証用ソフトウェアの概要は下記のとおりです．

▶ 内容確認

ICに書き込まれたデータを読み込み，その情報とPC内データベースに関連づけられている情報を一覧リストにして表示します．RFエンコーダにて書き込んだ情報と表面印字情報の検証を行うことができます．

▶ POS管理

ICに書き込まれた商品情報を読み込み，どの商品がいくつあり，合計金額がいくらかなのかを即座に計算します．従来のバーコードでは同じ商品がある場合，個数を確認しながら一つずつ計算することになりますが，同検証環境ではRFタグを使用した場合の作業効率の向上を体感していただくことができます．

▶ 入出庫管理

上記と同様に，ICに書き込まれた商品情報(コード)を読み込み，複数個を同時に入庫処理・出庫処理することができます．また画面上にはコード表示ではなく，商品名が表示されます．

▶ 勤怠管理

ICに書き込まれた個人情報(社員番号)を読み込み，出退勤情報(だれがいつ出勤・退勤したかなど)を管理/表示することができます．さらに，添付のユーザーズ・マニュアルにより，ソフトウェ

 4 通信範囲の切り分け

● 通信距離はどこまで必要か

RFIDシステムの導入を検討するときに，非接触認識およびアンチコリジョン機能[注1]に過度に期待するあまり，可能な限り通信距離を長く確保したい，あるいは一度に認識する数量を増やしたいという要望が非常に多く聞かれます．しかし，安易にその考え方で進めていった場合，以下の問題が発生することになります．

注1：アンチコリジョン機能とは，リーダ/ライタの読み取り可能範囲内に複数のタグが入ったときに，それぞれのタグを混信することなく正常に読み書きできる機能．

1) 読み取り条件不均一による認識精度の低下
2) 読み取りエラー対象の除去が困難
3) 想定していないタグの認識が発生

読み取りエラーが頻発したり，あるいは本来認識したくないものまで読み取ってしまったりということでは，RFタグを採用しておきながら作業性と精度がかえって低下することになってしまいます．

このような問題を未然に防ぐため，次のような点が重要です．

1) 必要以上の通信距離をとらない
　人手搬送の場合は15cm程度
2) 一括認識数量は，認識精度が確保でき，また迅速なエラー

表A　RFタグ「TS-L101」,「TS-L102」

製品名	TS-L101	TS-L102
タグ・サイズ	20×55mm	20×55mm
通信周波数	13.56MHz	13.56MHz
登載ICチップ	I-CODE 1	I-CODE SLI
ICメモリ容量	48バイト（ユーザ・エリア）	112バイト（ユーザ・エリア）
ICメモリ種類	64バイト EEPROM	128バイト EEPROM
使用基材	PET	PET
枚数	500枚/ロール	500枚/ロール

アの開発を行うことで，本格システムへのスムーズな移行も可能です．

なお，上記4種類のソフトウェアは，同梱のRFタグ「TS-L101」「TS-L102」を混在させても使用することができます．

タグおよび各機器のおもな仕様を表A～表Cに示します．

近年，さまざまな管理用途でのRFタグ/カードの試験導入が着実に増えてきています．同開発キットで，粘着ラベル・タイプRFタグ・システムのメリットである"使いやすさ"を幅広く訴求していくとともに，本格運用レベルでのシステム導入を促進し，RFID市場全体の一層の拡大に利用されると思います．

● 製品概要

▶ システム構成

- RFタグ「TS-L101」,「TS-L102」各1ロール（500枚／1ロール）
- RFエンコーダ搭載型プリンタ「DC-P301」1台
- リーダ/ライタ［（株）エフイーシー製］
- ソフトウェア（タグ発行用，検証用）CD-ROMで提供

▶ システム価格　980,000円/セット

▶ 製品に関するお問い合わせ先

リンテック（株）アドバンストマテリアルズ事業部門
情報通信材料部
〒112-0004　東京都文京区後楽2-1-2興和飯田橋ビル
TEL：03-3868-7737，FAX：03-3868-7726

表B　RFエンコーダ搭載型プリンタ「DC-P301」

製品名	DC-P301
印字方式	熱転写方式
印字密度	12ドット/mm
印字速度	50～200mm/s
バーコード	EAN/JAN13，EAN/JAN8，UPC/A，UPC/E，CODE39，NW-7，ITF，CODE128，CODE93，カスタマ・バーコード
二次元コード	PDF417，QRコード（MODEL2対応）
インターフェース	シリアル・インターフェース（RS-232C準拠）
装置サイズ	250（W）×380（D）×286（H）mm（突起物除く）
重量	12kg（本体）
電源	AC100～120V/AC200～240V（50/60Hz）
消費電力	最大420VA（12ドット）
RFID仕様	
エンコード	I-CODE 1，I-CODE SLI

※同機はソフトウェアの変更により，my-d，MN63Y1050にも対応可能．

表C　（株）エフイーシー製リーダ/ライタMRW-3010-Ant

製品名	MRW-3010-Ant
対応ICチップ	I-CODE 1，I-CODE SLI
通信周波数	13.56MHz
RF出力	最大500mW
ホスト・インターフェース	RS-232C（半二重通信，D-sub 9ピン）
装置サイズ	コントローラ：76（W）×130（D）×32（H）mm アンテナ：145（W）×145（D）×30（H）mm
重量	340g
電源	DC12V（専用ACアダプタから供給）φ2.1mmミニ・ジャック
消費電流	RFキャリアOFF時：110mA RFキャリアON時：260mA

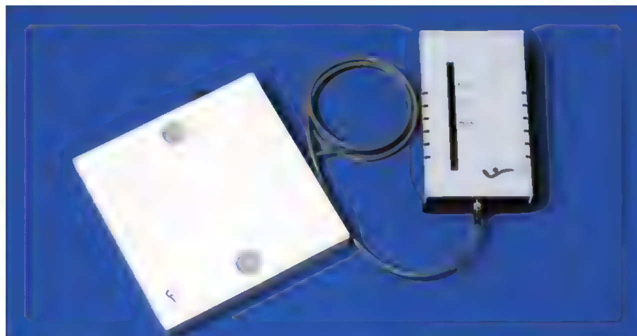


写真3　RFタグ・リーダ/ライタ MRW-3010-Ant

除去ができる範囲におさえる

人手搬送の場合は多くとも5〜10程度

1)は人手で運用するにあたって，「認識させる対象」と「認識させない対象」を効率よく切り分けようとした際の一つの目安です．かりに50cmの通信距離があった場合，ある対象物を通信範囲外に持ち出すために作業動線が必要以上に伸びてしまいます．

また，タグの最適サイズ20×55mmは，この通信距離を満たすアンテナ・サイズということも根拠としています．

2)はタグの読み取りエラーが発生した際，集合体の中から速やかにNG個体を抽出できうる数量といえます．かりに100個の集合体を通信範囲に置き，認識されたタグが98個であったとき，2個のNG個体を抽出するのがいかに困難かは明らかです．

5 リーダ/ライタ選定の指針

● 使いやすいリーダとは

ここまでで「粘着ラベル・タイプRFタグ」+「RFエンコーダ搭載型プリンタ」を軸とするシステムの利便性，優位性について述べてきましたが，RFタグ・システムの運用にあたっては，さらにリーダ/ライタ環境が必要となります．

リーダ/ライタに関しては，要求される性能や使用環境に合わせて，近接タイプから長距離タイプまでの定置式のもの，またはおもに機動性が必要な際に用いられるハンディ・タイプなどから，適切なものを選定することになります．

リーダ/ライタ選定基準の例としては，以下のような項目に注意する必要があるでしょう．

1) 機動性の有無
2) 通信距離
3) 装置サイズ
4) インターフェース
5) コスト
6) そのほか信頼性や運用のしやすさ

簡単に例を挙げただけでも多くの選定基準があることがわかります．とくに近年数多くの機種が発売されている中で，選定基準がついコストになりがちな面もありますが，ここでも重視したいのが「使いやすさ」とその結果として強化される「信頼性」です．

それでは，定置式リーダを選定する際の「使いやすさ」とはどのようなものでしょうか．その答の一つとして，「稼動時の状態表示がされているか」が挙げられます．

実際のシステム運用において，必ずしもリーダ/ライタの上位側にPCがあるとは限りません．システムの安定性を考え，マイコン・ボードを使用した制御を行うこともしばしばです．このような場合，リーダ/ライタの状態をPCの画面に表示することができないため，そのリーダ/ライタが通信を行っているのか，待機状態にあるのか，あるいは止まっているのかを知ることができません．つまりリーダ/ライタそのものに表現（表示）させる必要があるのです．前ページのコラムで紹介している開発キットでも採用している「MRW-3010-Ant」（写真3）はLEDを搭載し，その状態を表示します．

また，作業者が対象物（RFタグ）をアンテナ部分に近づけ，読み取らせる場合，その人の目は通常アンテナの方向を向いています．そして，その反応状態を確認することとなりますが，アンテナ自体と確認用の画面とが必ずしも同じ方向にあるとは限りません．むしろアンテナと画面は，作業者の立ち位置の角度から45度以上ずれていることのほうが多いといえます．作業性を考えた場合，物を近づける方向，つまりアンテナ自体に表示機能があることが望ましいのです．MRW-3010-Antのアンテナは，赤と緑のLEDを搭載しており，上位側からのコマンドで点灯制御を行うことで作業効率向上を実現します．

6 今後の課題

RFタグはその特有の機能から，各方面からその実用化が期待されていながら，最初に触れたように，さまざまな要因から普及が追いついていないといえます．

それがある部分で性能やコストに起因することは事実ですが，サービスを提供する側としては，「これまで本当に使いやすい環境がユーザに提供されていたのか」，また「本来的なRFタグの特徴にあった運用方法が提案されていたのか」を改めて考えていかなければなりません．

むらかみ・たかかず
リンテック（株）アドバンストマテリアルズ事業部門 情報通信材料グループ

Appendix

USB対応リーダ/ライタを使用した

Visual Basicによるアプリケーション開発

吉崎 辰美

はじめに

昨今,「RFID」や「無線タグ」,「非接触ICタグ」などといったことばが,毎日のように新聞紙上で賑わいをみせています.興味をもっている方は非常に多いのではないでしょうか.実際,身近にシステム化され,運用されているものとしては,JR東日本の「Suica」などがあります.使用してみて,その便利さを実感された方も少なくないでしょう.近年,ISOによる標準化や電波法の規制緩和などにより,このRFID業界へ多くの企業が参入してきました.その結果,比較的容易にシステム評価ができる簡易型の開発キットの価格が,個人でも購入できるレベルにまで下がってきています.また,アプリケーションを開発するためのツールも便利で使いやすくなってきており,個人でも趣味としてアプリケーションの開発を行っている人も多いのではないでしょうか.

そこで本稿では,非接触ICタグのアプリケーションを(株)エフイーシー製URWI-201とVisual Basic 6.0(以降VB6)を使用して作成する方法,とくにUID(チップ固有番号)を取得する方法に関して解説します.

USBリーダ/ライタの概要

URWI-201は,Windows上で動作するPhilips社(オランダ)製のI-CODE SLIおよびISO15693-3(カスタム・コマンドを除く)を対象としたUSB対応リーダ/ライタです.扱いが容易なアンテナ一体構造で,外形寸法が99×87×27mmと小型で低価格なタイプとなっています.また,オフィスなどにもマッチしたデザインで,パソコンの使用者の認証など,幅広い用途に使用することができます.

アプリケーション作成には,同リーダ/ライタを使用したPCアプリケーションの制作が容易なソフトウェア開発用キット(SDK:Software Development Kit)などで構成される,URWI-201開発キットを使用します.同キットは,非接触ICカード/タグに対して特別な知識がなくても簡単に操作を行うことができ,動作の確認や知識の習得ができる動作確認用のソフトウェアも同梱しています.



写真1 USB対応リーダ/ライタURWI-201(本体)

今回紹介する非接触ICタグのリーダ/ライタURWI-201は,USBのインターフェースにて通信を行います.以前のような通信ポートの設定やハードウェア制御という,ちょっとしたハードルを感じることなく,アプリケーション開発を行うことが可能です.

URWI-201開発キットの構成

今回使用する開発キットには,以下の内容のものが梱包されています.

1) URWI-201(本体):1台(写真1)
2) サンプル・タグ
3) CD-ROM:1枚
 ●ドキュメント(ユーザーズ・マニュアル)
 ●システム・ファイル
 ●DLLなど
 ●デモ・プログラム

以下,内容物について解説します.

1) URWI-201(本体)

USBインターフェースにてパソコンと通信を行います.また,USBから電源供給を行うので,外部にACアダプタなどの電源を必要としません.上部には3色のLEDがあり,通信の状況などをLEDの点灯により,知らせることが可能です.

2) サンプル・タグ

Philips社製I-CODE SLIを使用したタグを動作検証用に同梱しています.I-CODE SLIは,ISO-15693-3に準拠した非接触ICタグ用のチップです.112バイトのユーザ・メモリを有しており,簡単な情報をタグに持たせることも可能です.

3) CD-ROM

ユーザーズ・マニュアルや各種システム・ファイル,ソフトウェア開発キット(SDK)などが格納されています.DLLは,Windows 2000以降のバージョンに対応しています.

外観寸法を図1に示します.

CD-ROMに同梱される,URWI-201にアクセスするためのDLLを表1に示します.これらはアプリケーションをVB6にて作成する場合に必要です.

UID(固有ID)を読み取るアプリケーション開発

今回,URWI-201を使用してI-CODE SLIのUID(固有ID)を読み取るアプリケーションを実際に作成してみました.開発言語はVB6を使用しています.

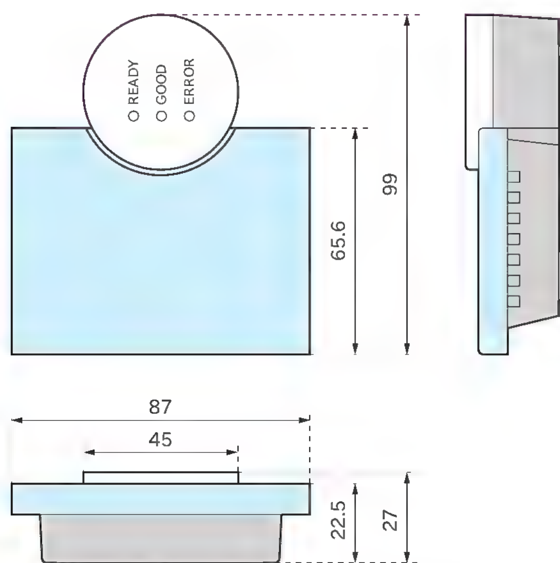


図1　USB対応リーダ/ライタ URWI-201の外観図

表1　DLLの内容

ファイル名	内　容
URWI2USB.dll	DLL本体
URWI2USB.bas	関数宣言ヘッダ・ファイル
status.bas	応答ステータス・ヘッダ・ファイル

表2　DLL関数一覧

コマンド	内　容	備考
URWI2_Initialize	DLLの初期化処理を行う	
URWI2_Terminate	DLLの終了処理を行う	
URWI2_GetRcvLength	最新レスポンス・データ長取得	
URWI2_GetRcvbuff	最新レスポンス・データ取得	
URWI2_ReadVer	R/Wバージョン・ナンバ取得	
URWI2_Inventory	ICカード・タグの衝突防止シーケンスの実行	必須
URWI2_StayQuiet	静止状態移行要求	必須
URWI2_ReadSingleBlock	単独ブロック読み込み実行	
URWI2_WriteSingleBlock	単独ブロック書き込み実行	

VB6起動後に，まずプロジェクトに標準モジュールとして関数宣言ヘッダ・ファイル「URWI2USB.bas」を追加してください．フォームにはコマンド・ボタンにより，「ReadVer」と「Inventory」を，オプション・ボタンにて「Flag」を選択させることとしました（図2）．

「ReadVer」はURWI-201のバージョン情報を得るためのコマンドで，「Inventory」はUIDを取得するためのコマンドになります（表2を参照）．大まかな手順としては，DLLのロード，Initialize[注1]，Inventory，…，Terminate，終了となります．

注1：Initializeは，アプリケーション実行時に一度だけ実行する．同様にTerminateは，アプリケーション終了時に一度だけ実行する．

図解すると図3のようになります．また，応答ステータス一覧を表3に示します．

「Flag」の選択は，以下のとおりです．AFIについてはここでの説明を割愛しますが，AFIを有効にすることにより，効率的にIDを取得するための設定を行うことができます．今回は，無効でも有効でも変化はありません．

動作指定フラグは次のとおりです．

0x06＝AFIは無効，衝突シーケンスは16スロット
0x16＝AFIは有効，衝突シーケンスは16スロット

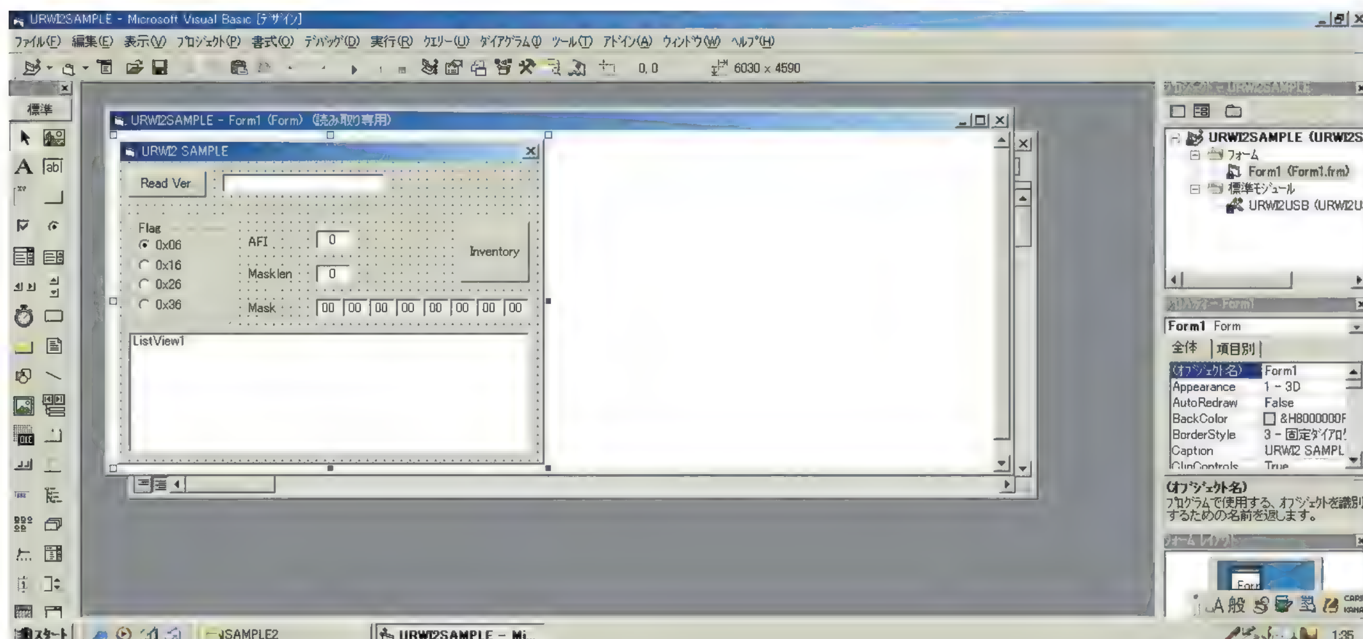


図2　VB6上でのフォームの編集

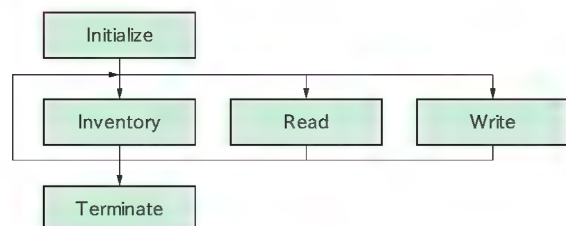


図3　処理の流れ

表3　応答ステータス一覧

ステータス	CODE	内　容
I2_OK	0	コマンド正常終了
I2_NO_TAG	1	ICカード・タグ応答なし
I2_CRCERR	2	ICカード・タグ通信CRCエラー
： （中略） ：		
ERR_COMM_ERROR	－114	USBポート・エラー

図4
UIDの内容

0バイト	1バイト	2バイト	3バイト	4バイト	5バイト	6バイト	7バイト
シリアル・ナンバ					TAG TYPE	IC Mfg.code	0xE0

表4　応答データ（1スロット指定時）

IpResp		
バイト数	内　容	備　考
1	衝突ビット位置	エラーの場合，このデータはない
1	フラグ	
1	取得したDSFID	
8	取得したUID（シリアル・ナンバ）番号	

表5　応答データ（16スロット指定時）

IpResp			
バイト数	内　容	範囲（値）	備　考
1	スロット1ステータス	XX	応答ステータス参照
1	スロット2ステータス	0	スロットのステータスがコマンド正常終了の場合
1	衝突ビット位置 注	XX	
1	フラグ	XX	
1	取得したDSFID	XX	
8	取得したUID（シリアルNo.）番号	XX	
： （中略） ：			
1	スロット15ステータス	応答ステータス参照	
1	スロット16ステータス	応答ステータス参照	

　`0x26`＝AFIは無効，衝突シーケンスは1スロット
　`0x36`＝AFIは有効，衝突シーケンスは1スロット

「衝突シーケンス」は複数枚のタグを読み込ませたときの動作を指定します．この値の1スロットか16スロットかの選択ですが，複数枚のタグを読み込ませる（IDを取得するなど）ときは，16スロットを選択します．1枚を読み込ませる場合は，1スロットを選択します．

応答データは表4，表5のとおりとなります．

UID（固有ID：シリアル・ナンバ）に関して

図4のように，UIDは8バイトで構成されています．0～4バイトはタグ識別子，5バイト目はタグの種類（I-CODE SLIは`0x01`）を示しています．6バイト目はIC製造者コード領域に割り当てられていて，たとえば，Philips社のタグは`0x04`，TI社のタグは`0x07`となっています．7バイト目は，`0xE0`で固定です．

以上から作成したプログラムのソース・リストをリスト1に示します．

実際にコンパイルしたソフトウェアでInventoryを行うとUIDを取得することが可能です．図5では，5スロットに「64 52 6C 12 00 00 07 E0」とUIDが表示されています．

タグはリード・オンリでもかまわない

RFIDのアプリケーションを開発する場合，UIDを使用してホストコンピュータのデータベースと連動させるのか，メモリに情報を持たせて管理するのかが，よく議論になります．一見タグに情報を持たせたほうが良いのではないかと思われる方が多いかと思います．

しかし，チップ破損時に備えたデータ・バックアップやそのタグの変更・更新を考えていくと，両者に大差はなくなってきます．UID

リスト 1　今回作成したプログラム

```
'
'   URWI2USB.dll 使用サンプル VB6
'
'   Copyright(c) 2004, FEC Inc.,Allright Reserved.
'
Option Explicit

Private hURW As Long

Private Sub Command1_Click()
    Dim flag As Byte
    Dim AFI As Byte
    Dim masklen As Byte
    Dim mask(7) As Byte

    Dim ii As Integer
    Dim ret As Long
    Dim Resp(1000) As Byte
    Dim rcvLen As Long

    Dim jj As Integer
    Dim ct As Integer
    Dim tmpstr As String

    Dim obItem As ListItem

    If Option1(0).Value = True Then
        flag = &H6
    ElseIf Option1(1).Value = True Then
        flag = &H16
    ElseIf Option1(2).Value = True Then
        flag = &H26
    ElseIf Option1(3).Value = True Then
        flag = &H36
    End If

    For ii = 0 To 7
        mask(ii) = CByte(Text3(ii).Text)
    Next ii

    If URWI2_Inventory(hURW, flag, AFI, masklen, mask(0),
                                Resp(0)) >= 0 Then
        ListView1.ListItems.Clear

        ct = 0
        If flag = &H6 Or flag = &H16 Then
            For jj = 0 To 15
                Set obItem = ListView1.ListItems.Add()

                'Status
                obItem.Text = CStr(Resp(ct))

                If Resp(ct) = 0 Then
                    'Bit
                    obItem.SubItems(1) = Hex(Resp(ct + 1))

                    'Flag
                    obItem.SubItems(2) = Hex(Resp(ct + 2))

                    'DSFID
                    obItem.SubItems(3) = Hex(Resp(ct + 3))

                    'UID
                    tmpstr = ""
                    For ii = 0 To 7
                        tmpstr = tmpstr & Format(Hex(Resp(ct
                                   + 4 + ii)), "#00") & " "
                    Next ii
                    obItem.SubItems(4) = tmpstr

                    ct = ct + 12
                Else
                    ct = ct + 1
                End If
            Next jj
        Else
            Set obItem = ListView1.ListItems.Add()

            'Status
            obItem.Text = CStr(Resp(ct))

            If Resp(ct) = 0 Then
                'Bit
                obItem.SubItems(1) = Hex(Resp(ct))

                'Flag
                obItem.SubItems(2) = Hex(Resp(ct + 1))

                'DSFID
                obItem.SubItems(3) = Hex(Resp(ct + 2))

                'UID
                tmpstr = ""
                For ii = 0 To 7
                   tmpstr = tmpstr & Format(Hex(Resp(ct + 3
                                        + ii)), "#00") & " "
                Next ii
                obItem.SubItems(4) = tmpstr

            End If
        End If
    End If
End Sub

Private Sub Command2_Click()
    Dim Resp(255) As Byte
    Dim ver As String
    Dim rcvct As Long
    Dim ii As Integer

    If URWI2_ReadVer(hURW, Resp(0)) = 0 Then
        rcvct = URWI2_GetRcvLength(hURW)
        For ii = 0 To rcvct - 1
            ver = ver & Chr(Resp(ii))
        Next ii
        Text4.Text = ver
    End If
End Sub

Private Sub Form_Load()
    URWI2_Initialize hURW

End Sub

Private Sub Form_Unload(Cancel As Integer)
    URWI2_Terminate hURW
End Sub
```

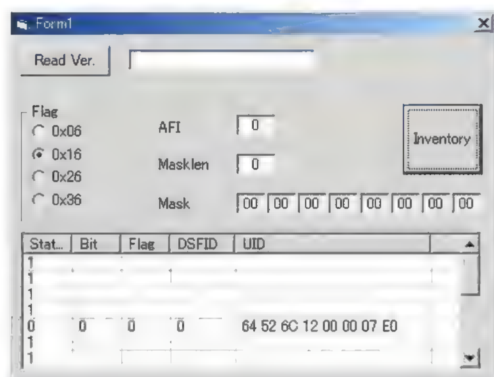


図 5
読み取った
UID の表示

がリード・オンリであれば，今回紹介した程度で，アプリケーションを開発することができます（当然，データベースの知識は別途必要になる）．読者の方も一度試してみてはいかがでしょうか．ビデオ・テープの管理や会社のファイル管理などの用途で，便利に使用できると考えています．

よしざき・たつみ　（株）エフイーシー 営業部

アンテナとタグの間の通信を視覚化して把握する

3 回路・システム・3D 電磁界シミュレータによる RFID 設計

門田 和博

RFタグには，ディジタル・データをやり取りするというディジタル的な側面と，アンテナとコイルの間を電磁波でやり取りするというアナログ的な側面がある．また，リーダ/ライタとタグ間の距離と周波数によって使用される信号伝搬も異なる．
そこで本章では，リーダ/ライタとタグの間の通信をシミュレータを用いることにより，目に見えない電磁波の影響を視覚化して客観的に把握しつつ設計する手法について解説する．（編集部）

はじめに

RFタグの設計において通信特性を左右するアンテナの設計は電磁界解析ツールを用いることで容易にその現象を把握することが可能である．本章では3D電磁界シミュレータを用いたアンテナ設計と，回路とシステム・シミュレータとの連携による送受信特性解析について紹介する．

1 13.56MHz RFタグ用アンテナ

● アンテナの概要

図1のようなRFIDカードでは，エッチングやスクリーン印刷などによりアンテナ・コイルと呼ばれるパターンを形成している．このアンテナ・コイルは同じくパターンで作成されたコンデンサおよび接続される回路側のコンデンサとの共振を，13.56MHzに同調させることでリード/ライトを可能とするためのものである．

ここで，"アンテナ・コイルと呼ばれる"という表現を使ったのには理由がある．実はリーダ/ライタとタグ間の距離によって通信（信号伝搬）の方法が異なるからである．

通信方法は距離によって以下の方法に分かれる．

- リーダ/ライタとタグ間が1/2波長以下の場合
  →誘導磁界による通信
- リーダ/ライタとタグ間が1/2波長以上の場合
  →電波による通信

したがって，13.56MHzの1波長が22mほどであることから，近接型RFIDシステムでは携帯電話のような無線通信用アンテナの動作とは異なり，コイルの結合により通信しているということである．

● 等価回路による考察

ここでRFタグの等価回路を考えてみる．図2はRFタグの等価回路で，

L：インダクタンス
R：直列抵抗
C：平行平板間のキャパシタンス

である．

この回路の共振周波数F_0はトムソンの式，

$$F_0 = 1/2\pi\sqrt{LC}$$

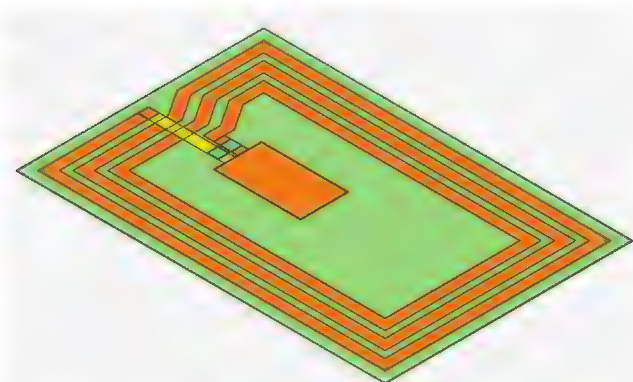


図1　RFIDカード内のアンテナ・パターン

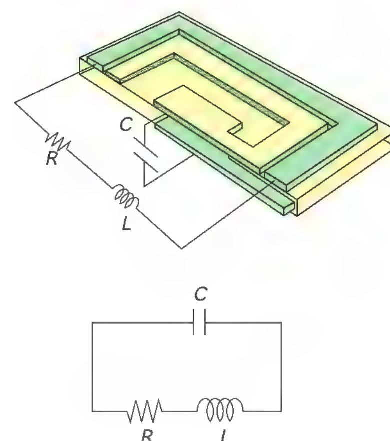


図2
RFタグの等価回路

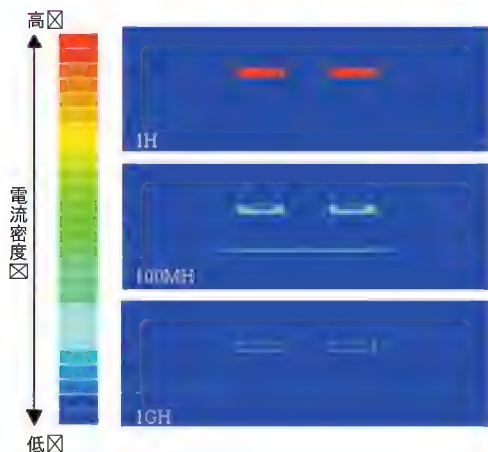


図3　各周波数における導体断面の電流密度分布

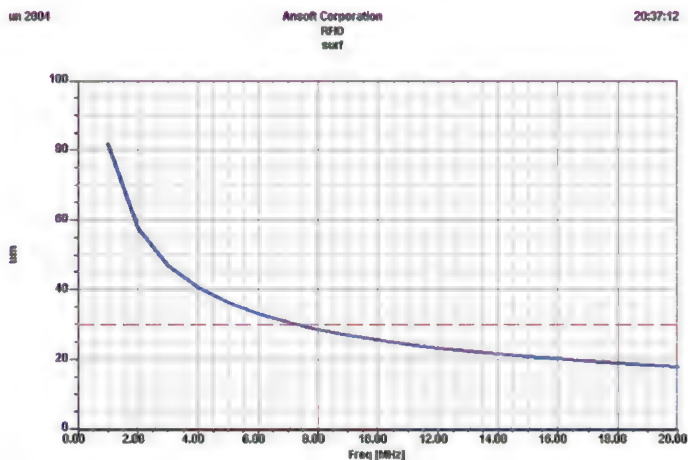


図4　表皮厚さの周波数特性

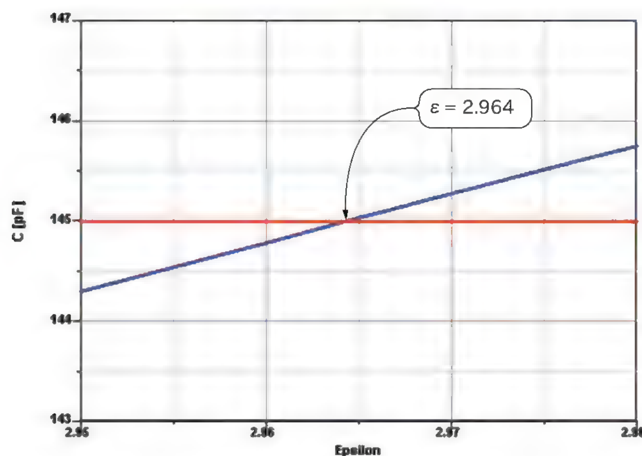


図5　実効誘電率の解析

で表され，Q（Quality Factor）は

Q= 蓄積エネルギ/損失エネルギ

で表される．

ただし，上記の式から導出される結果は理想状態での結果であり，物理的な要素はまだ含んでいない．そのため，通常は，この理想状態から得た結果を元に試作して，後は実機で調整して合わせ込むという作業が必要になる．しかし，3D電磁界解析ツールを使うと，試作することなくRFIDの設計を物理的な要素を含めて調整を行うことができる．

電磁界解析をするときの留意点として，表皮効果の考慮がある．導体中を流れる信号の周波数が高くなると，電流は導体表面に集中する．表皮の厚さは以下の式で求められる．

$$\delta = \sqrt{2/(2\pi \cdot f \cdot \sigma \cdot \mu_0 \cdot \mu_r)}$$

f：信号の周波数

σ：導体の導電率（S/m）

μ_r：導体の比透磁率（通常の金属では1）

μ_0：真空中の透磁率 $4\pi \times 10^{-7}$ Web/Am

この式より，導電率：3.8×10^7 S/mのアルミニウムの13.56MHzにおける表皮厚さは22μmとなった．

図3は幅：100μm，厚さ：20μmのマイクロストリップ線路の各周波数における電流密度分布のようすである．周波数が高くなると導体表面のみに電流が流れていることがわかる．

図4に30μm厚アルミニウム導体の表皮厚さの周波数特性を示す．この表からわかるとおり，8MHzあたりから導体厚みより表皮厚みのほうが薄くなってくる．したがって，13.56MHzにおいては表皮厚みを考慮した解析が必要となるのである．

設計上もう一つ重要なのが，タグに使用する誘電体材料の実効誘電率である．カタログに記載されている誘電率と実際に使用する周波数での誘電率は必ずしも一致しないので，実効誘電率は実際に容量パターンを作成してそのキャパシタンスを測定した結果と，電磁界解析上で同一のモデルを作成して基板の誘電率を変化させたときのキャパシタンスの値が一致する誘電率を求める必要がある．

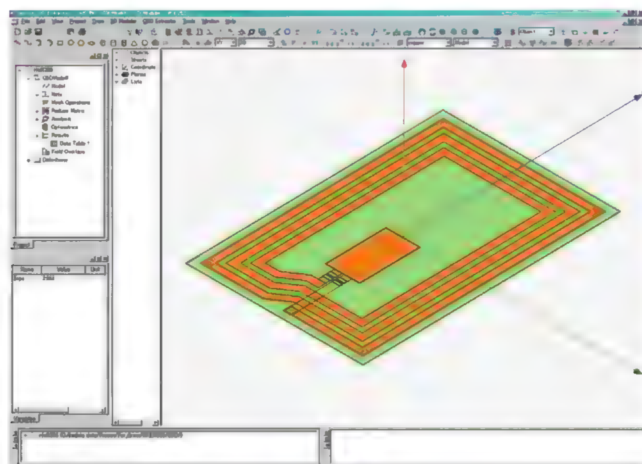
例として図1のタグ・モデル内の容量パターン部分を用いた結果を図5に示す．実測容量は145pFで，電磁界解析結果の容量値が一致するのは誘電率ε= 2.964となった．したがって電磁界解析で用いる実効誘電率にはこの値を用いれば，より実測に近い解析ができることになる．

● 電磁界シミュレータによる検証

これらの留意点を考慮したうえで設計したタグのパターンを図6の3D電磁界LCR抽出ツールのQ3D Extractorを用いてLCRを抽出してみた．結果はL=974.6nH，C=145.5pF，R=338mΩとなり，LCの値からトムソンの式で共振周波数を求めてみると，F_0=13.36MHzとなる．

また同じ3Dモデルを高周波電磁界シミュレータHFSSを使用して電磁界解析することにより図7のような，タグのSパラメータ（高周波でよく使われるインピーダンスの周波数特性を示すパラメータ）およびQ値を求めることができる．

Information —— TDK，高速無線通信方式UWBの通信機器の評価・測定サービスを開始
TDK（株）は，高速無線通信方式UWB（Ultra Wide Band）通信機器の評価・測定サービスを開始した．UWBは，3.1GHzから10.6GHzの周波数帯域を利用し，3mの距離において最大480Mbpsで通信できる通信方式．


図6　Q3D Extractor（Ansoft社）

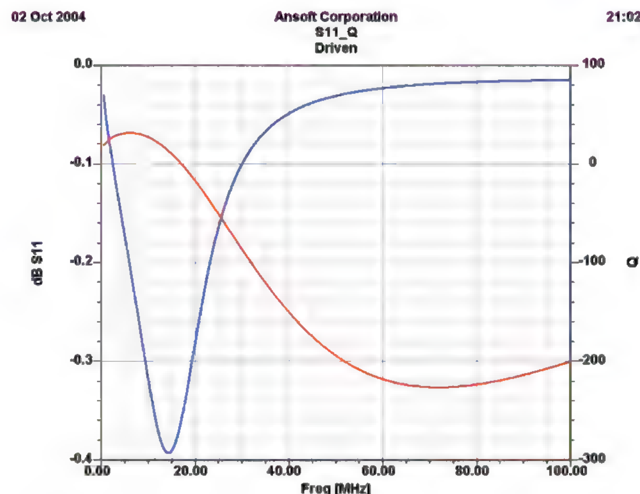


図7　HFSS（Ansoft社）によるSパラメータおよびQ値解析結果

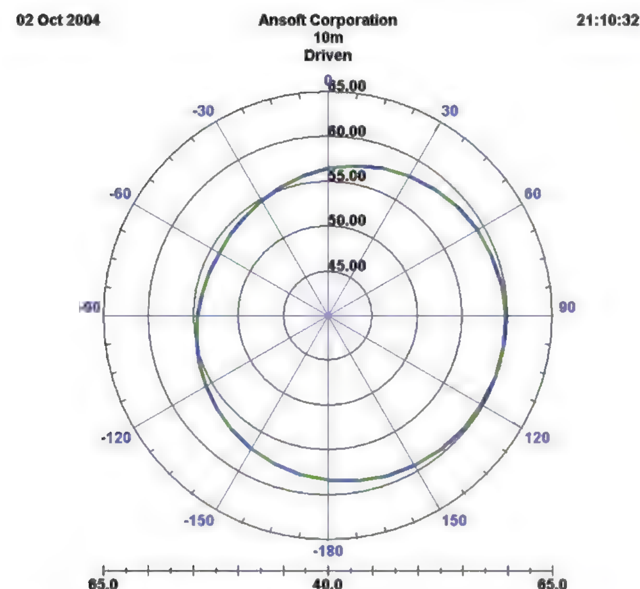
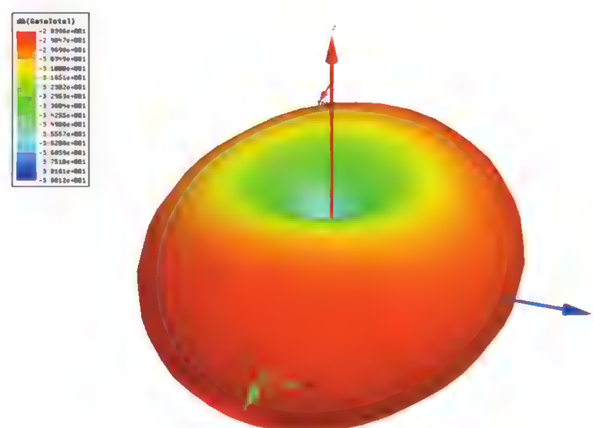


図8　10m遠場電磁界強度（dBμV/m）


図9　3Dでの遠方界の表示

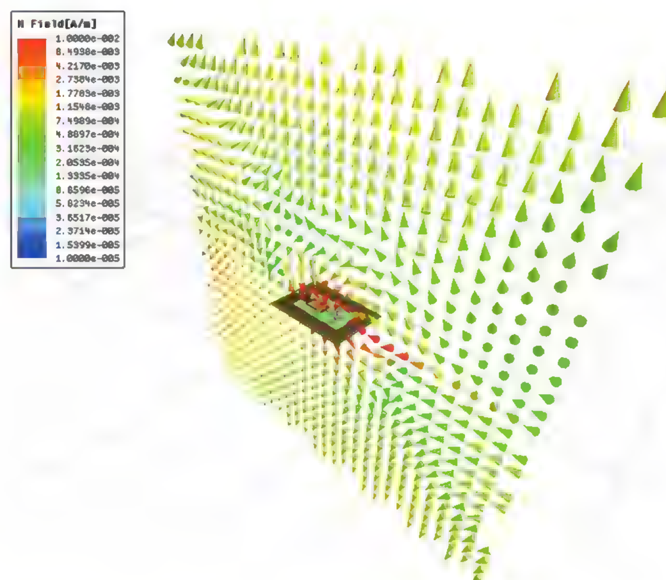


図10　近傍磁界の表示

　また，このアンテナの10m遠場電界強度（dBμV/m）および遠方界の3D表示，近傍磁界を図8，図9，図10のように解析することにより初期設計段階でアンテナの放射特性を把握し，システムから要求される特性を試作することなく検討することが可能となる．

2 リーダ/ライタ用アンテナ

● アンテナ形状の決定

　リーダ/ライタ用のアンテナにはループ・コイルが用いられる．ループ・アンテナの磁界の強さはループ半径Rと垂直方向の距離Zが等しいところで最大値をとるが，ループ半径Rが大きくなるとループの中心から垂直方向の磁界の強さは減少するようになるため，リーダ/ライタにどのような指向性を持たせ

1
2
App
3
4
5
6

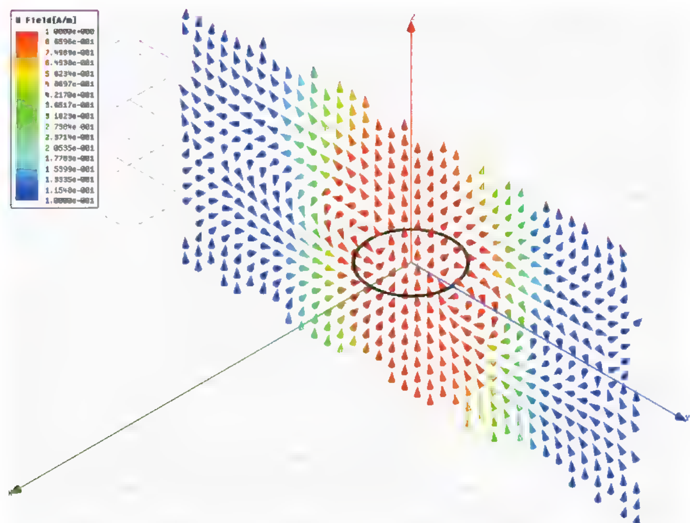


図 11　半径 74mm ループ・アンテナの磁界強度

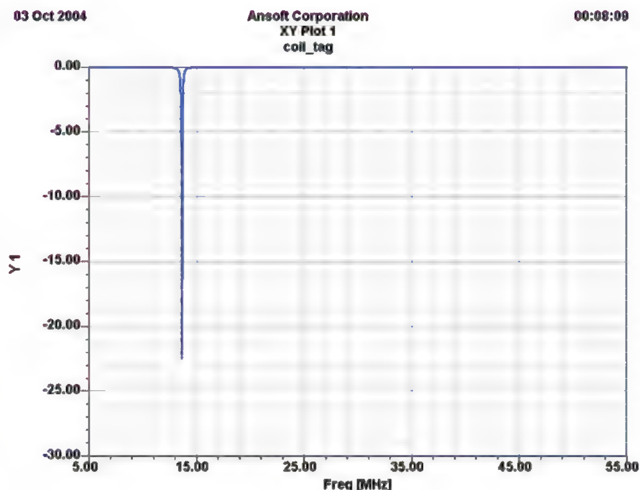


図 12　ループ・アンテナの S パラメータ(S11)

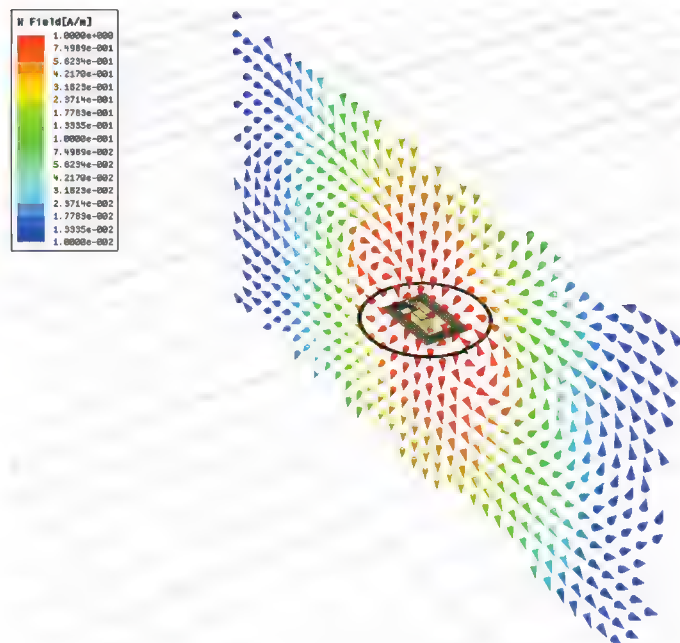


図 14　リーダ/ライタとタグ間の磁気結合のようす

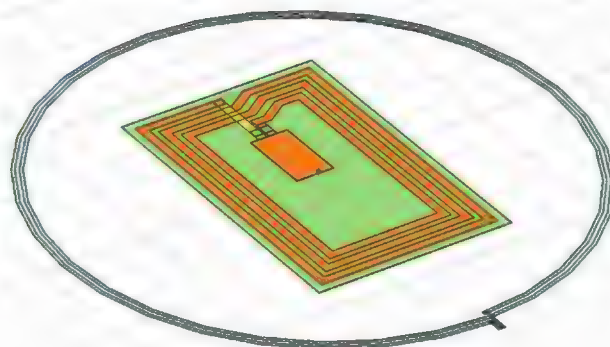


図 13　リーダ/ライタ用アンテナ+タグ・モデル

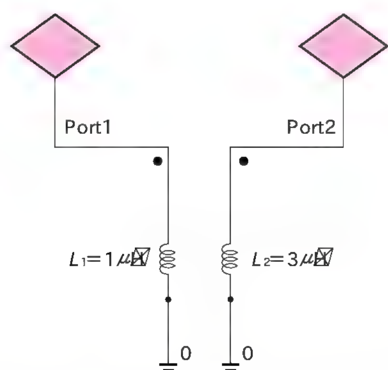


図 15　トランスの回路シミュレーション

るかによってループ半径 R を決定する必要がある．

図 11 は半径 74mm のループ・アンテナの磁界強度の解析結果である．このアンテナもタグと同様にループ・コイルとコンデンサで構成されており，その周波数特性は図 12 のとおり 13.56MHz で共振していることがわかる．

3 リーダ/ライタとタグ間伝送特性の解析

13.56MHz RFID では解析周波数に対して通信距離が短いため，リーダ/ライタ用アンテナとタグを同一の電磁界解析モデルとすることで，アンテナ間の伝送特性を検証することができる．

図 14 は図 13 のモデルでリーダ/ライタとタグとの磁気結合のようすを磁界の強度で示したものである．

ここで，この二つのアンテナの等価回路を考えてみる．二つのアンテナ・コイルが磁界結合していることから，トランス回路になっていることがわかる．

図 15 のトランス回路で L_1= 1μH がタグ，L_2= 3μH がリーダ/ライタとすると，回路シミュレーションでは結合係数 k でコイル間の結合を表現して解析することになるが，この結合係数

kはコイル間の相互インダクタンスMから$k=M/\sqrt{L_1 \cdot L_2}$で算出することができる．

では相互インダクタンスMはどのように求めるのか？ これは実測でL_1とL_2を順接続（L_1とL_2に流れる電流が逆向き）したときのインダクタンス値L_aと逆接続（L_1とL_2に流れる電流の向きが同じ）したときのインダクタンス値L_bから，

$$M=(L_a-L_b)/2$$

で算出することができる．

実測が可能な場合は上記の計算によりリーダ/ライタとタグ間の結合を把握することはできるが，設計の初期段階で電磁界解析を用いれば結合の状態を周波数特性も含め，アンテナ-コイル間の距離を変えたときの伝送特性を試作前に把握することができる．これは図16のように，タグの垂直方向の位置を変数として解析することで容易に行うことができるのである．

この結果を図17に示す．この図からわかるとおり，アンテナ-コイル間の距離により相互インダクタンスが変化するため，アンテナの共振周波数も変化する．

このように，試作をしなければ把握できないアンテナ間の伝送特性を設計の初期段階で電磁界解析ツールを用いることにより，最適なアンテナ特性を検証することができる．また，ここでは紹介しないが，磁界制御のためのリーダ/ライタ側に設置するフェライト材料の効果や金属板の影響などを含めた検討も可能である．

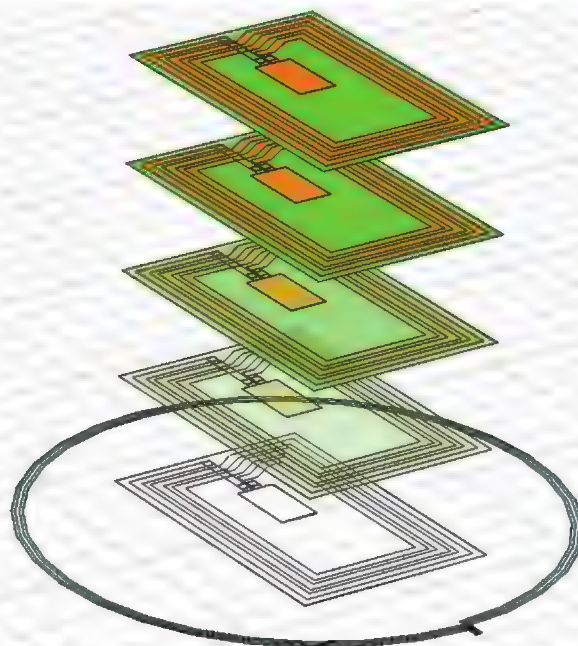


図16　タグ位置を変数とした電磁界解析

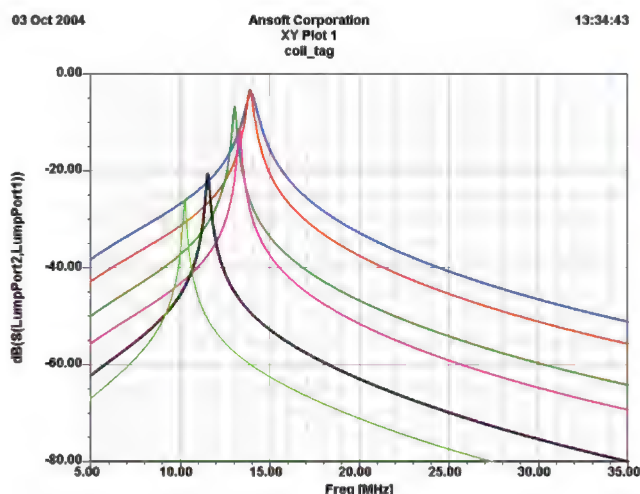


図17　タグの距離を変化させたときの伝送特性

4 リーダ/ライタとタグのアナログ回路

タグのアナログ回路は信号処理回路およびメモリとともにLSI化されているため，タグの設計においては回路設計を行うことはないと思う．したがって，RFIDに使われるアナログ回路設計でポイントとなるのは前述のアンテナ周りになる．ここではそのほかアナログ回路部分について要点のみ記す．

近接型RFタグでは電源を持たず，リーダ/ライタから得る磁界のエネルギを整流してICチップを駆動するための直流電源を得ている．しかし，リーダ/ライタから得られる磁界のエネルギは前述のとおり距離によって異なるため，単純に整流しただけでは一定の電圧を確保できない．

では，どうするのか？　簡単な方法としてはツェナ・ダイオードを用いてICを駆動する電圧以上にならないようにする方法がある．ツェナ・ダイオードはそのデバイスのもつツェナ電圧V_z以上の電圧がかかると抵抗値が下がり，ICを破壊するような高電圧がかかるのを阻止する働きをするのである．しかしながら，これは高電圧には対応するがIC駆動電圧より低い整流電圧の場合，電圧を増幅してくれるわけではないので，タグのアンテナから得た磁界エネルギが小さいと，電源電圧が不足しICが駆動できない状態になるということである．よって，13.56MHzの近接型RFIDの通信可能距離は携帯電話などとは異なり，アンテナのQ値と磁界エネルギからの電源再生能力に左右されるということになる．

図18に簡単なタグ電源供給回路を示す．ここでBPSK Dataとある部分はリーダ/ライタからの搬送波を16分周した副搬送波847kHzをリーダ/ライタへ伝送するディジタル信号でBPSK変調した信号である．R_1をこの信号でON/OFFすることにより搬送波にASK変調してリーダ/ライタへ伝送する．

5 リーダ/ライタとタグ間の通信性能シミュレーション

前述のアンテナ間通信特性の電磁界解析結果を用いてシステ

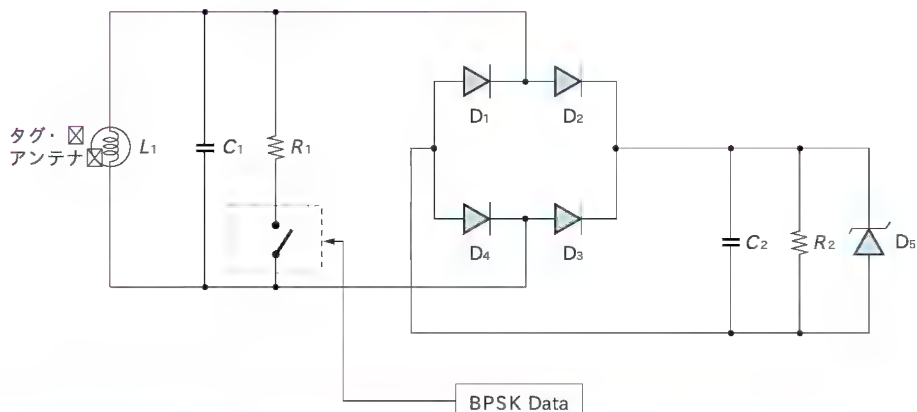


図18
簡単なタグのIC用電源供給回路

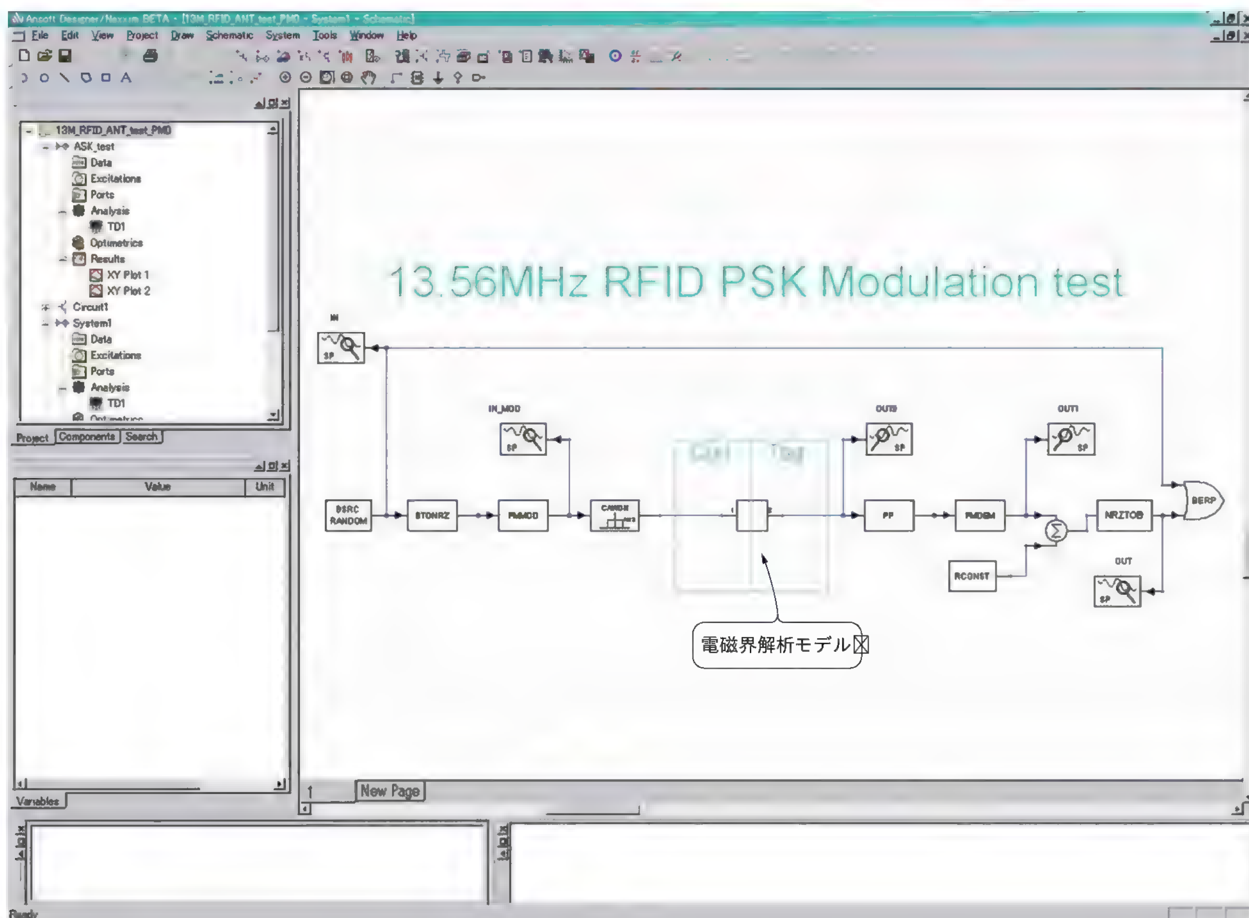


図19　Ansoft Designer（Ansoft社）でのシステム・シミュレーション

ム・シミュレータで通信性能を検証してみる．

13.56MHz RFIDでは信号を搬送波にディジタル変調（ASK，FSK，PSK）して伝送する．このようすをシステム・シミュレータで先に電磁界解析したアンテナ間の伝送特性と協調解析することで，仮想試作を行うことができる．

簡単な例としてリーダ/ライタ側から13.56MHzの搬送波をPSK変調して送信し，それをタグで受信しているようすを解析してみる．ここでは図19に示す3D電磁界解析ツールとの協調解析が可能な無線通信回路およびシステム解析ツールのAnsoft Designerを用いる．

このシミュレーションでは，伝送するディジタル信号として疑似ランダム信号を用い，13.56MHzの搬送波を疑似ランダム信号でPSK変調した後，リーダ/ライタのアンテナからタグのアンテナに伝送（図19中央のブラック・ボックスが3D電磁界

New Products――パナソニック コミュニケーションズ，ファイル棚，商品棚用のRFIDアンテナ・ユニットを開発
パナソニック コミュニケーションズ（株）は，ファイル棚や商品棚用のRFIDアンテナ・ユニットとして，基本アンテナ・ユニットと拡張アンテナ・ユニットを開発した．今後も開発を続け，RFIDリーダ/ライタとして2005年に商品化する予定．

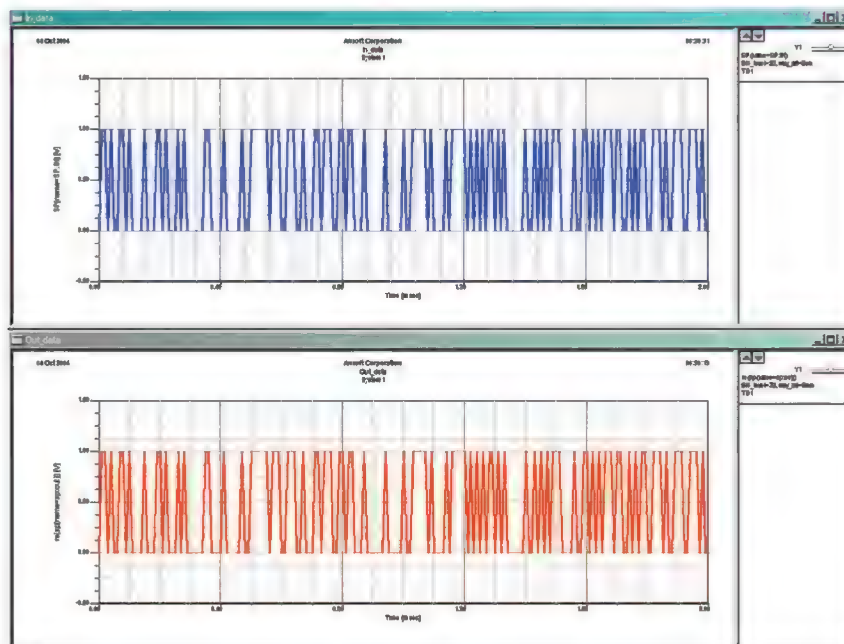


図 21
変調データと復調データの時間軸での比較

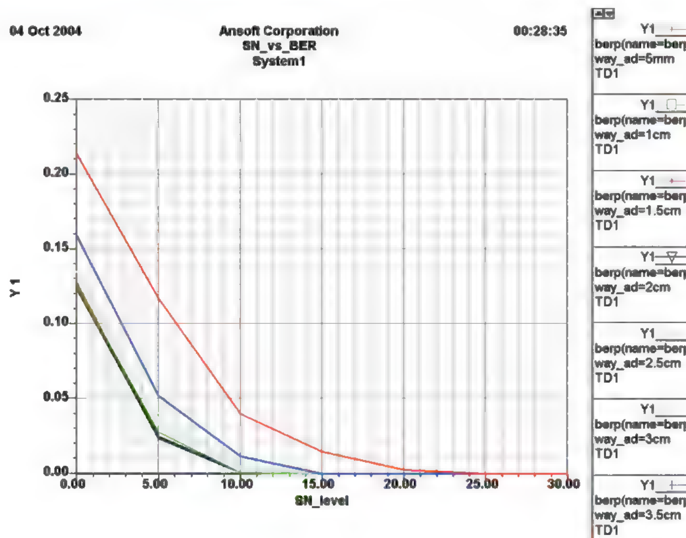


図 20　アンテナ間 vs ビット・エラー・レート特性

解析モデルにリンク），タグに伝送された信号を復調してビット・エラー・レートを計算している．

また，アンテナの前にノイズ源を配置しているので先に解析済みのアンテナ間距離の変動と合わせて，*S/N* 比の影響についても解析できるような設定になっている．

図 20 に示しているのがアンテナ間距離とビット・エラー・レートの関係で，*S/N* 比を横軸に取り縦軸をビット・エラー・レートとして，アンテナ間距離を変えたときの変化を表している．また，図 21 は搬送波に変調したデータとアンテナを伝送して復調されたデータのトランジェント解析結果である．

このように，システム・シミュレータと 3D 電磁界解析ツールで協調解析を行うことで，試作前に仮想試作を行うことができる．また，Ansoft Designer にはシステム，回路いずれにもチューニング，最適化，統計解析機能をもっているので，設計段階で物理形状と電子回路部を合わせて最適な設計をすることが可能である．

5 今後の RFID 開発へ向けて

現在各社で次世代の超小型タグの実用化に向けてマイクロ波 RFID の検討が進められている．日本では 2.4GHz 帯での実用化が進められている．2.45GHz の RFID は，13.56MHz と比較して波長が短くなる分だけ小型化が期待できるが，波長が短いゆえの問題点もある．アンテナについても前述したとおり，誘導磁界での通信から電波での通信の領域に入ってくるため，高周波の世界の経験者でないと敷居が高い分野になると思われる．

技術的に難易度が増すうえに，競争の激しい RFID 業界で勝ち残っていくにはさまざまな設計ツールをうまく使いこなして設計効率を上げ，試作回数を削減していくことが今後の RFID 開発のポイントとなることが予想される．

参考文献
(1) 苅部 浩；トコトンやさしい 非接触ICカード，日刊工業新聞社
(2) 根日屋英之，植竹古都美；ユビキタス無線工学と微細 RFID，東京電機大学出版局

かどた・かずひろ　アンソフト・ジャパン(株)

T-Engineフォーラムのユビキタス IDを使った

4 RFIDのセキュリティとプライバシ

尾立 英基/小林 真輔

これまでの章で解説したとおり，RFIDは非接触でさまざまな情報を得ることができる便利な技術だ．しかし，それは同時にRFIDを付けている人や物の情報を第三者が収集できるという危険をはらんでいる．収集したRFIDの情報をもとに，だれがどのような物をもっているか，だれがどこへ行ったのか…．RFIDを悪用すればそのようなことを調べられるようになる．RFIDの悪用を避けるためには，通信経路の暗号化や耐タンパ性などによるセキュリティの確保とシステム面でのプライバシに対する配慮が必要になる．

そこで本章では，T-Engineフォーラムが提唱するRFID,「ユビキタスID」を例に，ユビキタスIDの概要を述べたあと，RFIDにおけるセキュリティとプライバシの問題について考える．最後に，昨年ユビキタスIDを用いて行われた食品トレーサビリティ・システムの実証実験についても解説する．（編集部）

ここ数年で「ユビキタス・コンピューティング」ということばが多く使われるようになってきました．本章では，そのようなユビキタス・コンピューティングを実現するうえでの基盤技術の一つである「ユビキタスID」技術について解説します．

また，ユビキタスIDを用いた例として，食品トレーサビリティについての実験を紹介します．

1 ユビキタス・コンピューティングがもたらす世界

ユビキタスIDの説明を行う前に，背景となるユビキタス・コンピューティングの話を先に述べていきます.「ユビキタス」ということばは,「どこにでもある，遍在する」という意味のラテン語からきています．つまり，ユビキタス・コンピューティングとは，どこででもコンピュータが使えるようなコンピューティング・モデルのことを指します．ただし，今現在，多くの人が携帯電話などに代表される小型端末を持っていますが，これらのことを指すわけではありません．



図1　ユビキタス・コンピューティングがもたらす世界

あらゆるところにネットワーク通信可能な超小型のコンピュータを埋め込み，実世界を認識したコンピュータ群が協調して人間の生活をサポートするしくみのことをユビキタス・コンピューティングと呼びます．ユビキタス・コンピューティングの世界では，人間はコンピュータを意識することなく，さまざまなサービスを享受することができます．

ユビキタス・コンピューティングの本質は，言い換えるならば，身の周りのモノや人の位置や空間の把握，属性の管理をコンピュータが認識して，生活空間の中で見えないようにサービスを提供することといえます．このようなことが実現された場合，さまざまなことができるようになります．

たとえば，薬瓶に超小型のチップを付けることでコンピュータが薬瓶を認識できるようになり，認識した結果を使って薬の飲みあわせを教えてくれる，衣服に超小型チップを付けることで，自動的に洗濯機が洗い方を判断して洗ってくれる，冷蔵庫の中身が自動的に表示される，初めて訪れた町の地下街で，通路や壁に埋め込まれたチップの情報を読みとることで経路情報やおすすめの店を教えてくれるなど，さまざまなことができるようになります（図1）．

2 ユビキタスIDとは何か

このようなことを実現するうえで必要となる要素技術として，「ユビキタスID」があります.「ユビキタスID」技術は，ID（識別子）をあらゆるモノに付け，IDを用いてさまざまな情報を使ったサービスを提供する技術を指します．IDをコンピュータが読みとることでコンテキスト・アウェア（context aware）な処理を行うことができるようになります．ユビキタスIDタグ（ucodeタグ）は，モノを認識するためのIDです．現在，バーコードなどで使われているJANコードやISBNコードはモノの

種類に対して番号を付けていますが，ユビキタスIDタグはモノの一つ一つに別のIDを付ける点で異なっています．

次にIDがどのような形で用いられるのかを説明します．

前の項目で述べたようなサービスを提供するうえでは，さまざまなモノの情報が必要となりますが，それらの情報をすべてタグの中に格納しておくのではなく，タグから取得したIDを使って情報を検索し，その結果を使ってサービスを提供します．このようなユビキタスIDのアーキテクチャを図2に示します．まず最初にIDリーダを用いて，モノに付けられたユビキタスIDタグからIDを読み出します．読み出したIDを使って，ID管理サーバ（ユビキタスIDセンター，後述）に情報が格納されている場所を問い合わせます．次に情報を取得します．

このような方式を考えた場合，IDをどのように割り当てるか，どのように管理するか，IDに付随するデータをどのように扱うかといった問題が考えられます．

以降ではそれぞれについて考えていきます．

● ID空間

このように一つ一つのモノにIDを割り当てることを考えた場合，既存のID体系をどう扱うか，ID空間をどれくらいの大きさに定めるかということが問題になってきます．

▶既存のID体系（JANコードやISBN）などはどのように扱えばよいのか

まず最初の点ですが，既存のID体系を無視して，まったく新しいID体系を作り上げるのは，それはそれでやり方の一つだと思います．ただし，既存の体系を無視してしまうと無用な混乱を招くことになりかねません．そこで，既存の体系を包含する識別子体系として「メタID」をユビキタスIDでは採用しています．

「メタID」とは，ID体系の中のある領域に既存のID体系をそのまま取り込むことで実現したID体系のことを指します．IDを解釈するときに，ある部分のビット列を判定するだけで，そのIDが既存のID体系を含んでいるかどうかがわかるようになります．

▶モノをすべて認識するうえで，どれくらいのID空間があれば十分なのか

次に考えられる問題点ですが，どれくらいのID空間があれば十分でしょうか．すべてのモノに付けるわけですから，モノよりも多くのID空間が必要になります．また，IDを管理するうえで重要なのが，「IDを再利用しない」ということです．これはどういうことかというと，IDを管理するうえでIDの再利用を許してしまうと，どのIDをどこでどのように使われているかが管理できなくなってしまいます．このように考えると，IDは使い捨てで割り当てても十分な量の空間が必要になるといえます．

さらに先程述べたようなメタIDであるためには，ある程度空間に冗長性が必要になります．ただし，既存のID体系では可変長に設定されている場合がありますが，RFIDなどの技術を考慮すると，どの長さのIDでもよいというわけではなく，固定長のほうが望ましいことがあります．そのため，可変長は基本仕様では含まないことにしています．

それでは，実際にID長がどれくらいあれば十分かを考えます．ucodeタグは，128ビットあれば基本的には十分であると考えています．この128ビットという数値ですが，2^{128}を10進表現すると次のようになります．

$$2^{128} = 3.40 \times 10^{38} = \text{約}340\text{澗（かん）}$$

この340澗という数値は非常に大きな数値です（日常生活で「澗」という単位を筆者は使ったことはない）．そこで参考までに次のことを考えてみます．たとえば，地球上$1mm^2$あたりにIDを割り当てたとして，どれくらいのIDを割り当てられるかを考えてみます．地球の表面積は約$5.1 \times 10^{14} m^2$です．すると，およそ700,000,000,000,000,000個（約70京個）のIDを割り当て

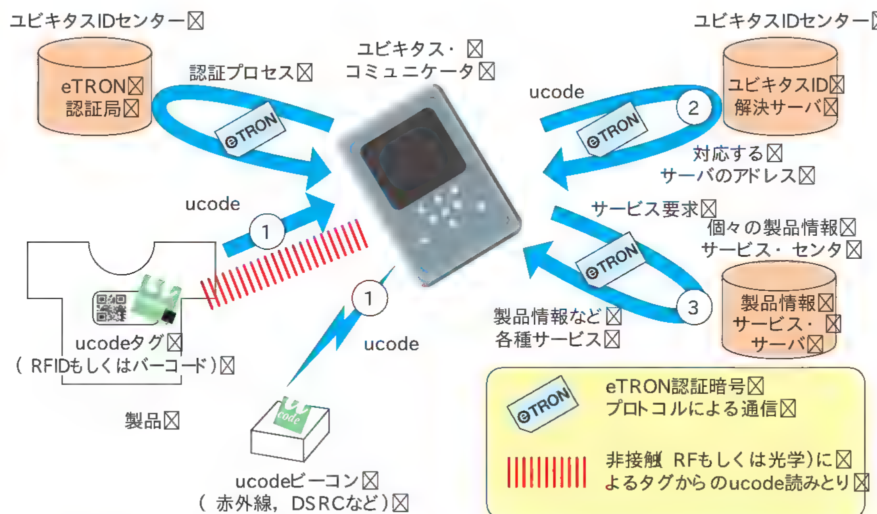


図2　ユビキタスIDのアーキテクチャ

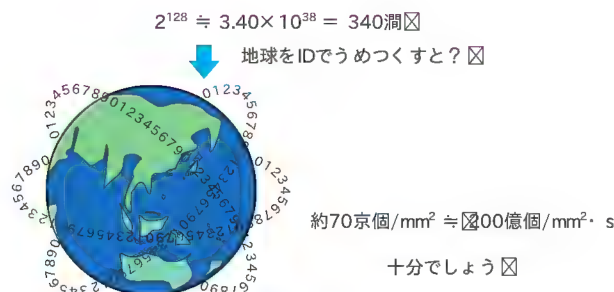


図3　IDは128ビットで十分か？

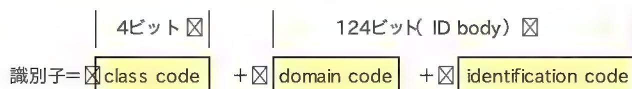


図4　基本IDの構成

表1　定義済みクラス・コード

クラス・コード	クラス名称	識別子本体部の内容
0000～0111	予約	
1000	ClassA	dc＝12ビット，ic＝112ビットのid
1001	ClassB	dc＝28ビット，ic＝96ビットのid
1010	ClassC	dc＝44ビット，ic＝80ビットのid
1011	ClassD	dc＝60ビット，ic＝64ビットのid
1100	ClassE	dc＝76ビット，ic＝48ビットのid
1101	ClassF	dc＝92ビット，ic＝32ビットのid
1110～1111	予約	

cc(class B)				dc(28ビット)	id(96ビット)	
0			3	4　31	32　83	84　127
1	0	0	1	JANドメイン識別子	JANコード	製品個別ID

図5　ucodeにおけるJANコードの割り当て

ることができることになります．これは，全地球上で1mm²あたり1秒間に1IDずつ消費していったと考えた場合でも，全ID空間を消費するまでに約20,000,000,000年(約200億年)かかる計算になります(図3)．

以上のように考察すると，128ビットで基本的には十分であることがわかると思います．

● 基本IDの構成

基本IDの構成を図4に示します．基本コードは，4ビットのクラス・コード(class code)と124ビットのID本体(ID body)から構成されます．さらにID本体は，ドメイン・コード(domain code: dc)と識別コード(identification code: ic)から構成されます．ここで，ドメイン・コード部分は既存コードを包含する場合の識別子となり，識別コードの部分は，既存のコードがそのまま入ることになります．

クラス・コードに関して，もう少し詳しく説明します．クラス・コードは，下位124ビットのID本体のコード・クラスを表現します．現在クラス・コードは，AからFまでの6種類規定されています(表1)．それぞれのクラス・コードに対応して，ID本体のdcとicのビット長が決定されます．クラスの選択は，既存のID体系を取り込む場合は，既存のID体系のビット長がicのビット長に収まるようにクラスを選択します．そうでない場合は，識別する必要があるデータの数やドメインの種類に応じて決定します．

既存のIDをucodeに割り当てる場合を考えてみましょう．例としてJANコードの割り当て例を示します．JANコードはバーコードなどでもっとも多く用いられているコード体系です．JANコードは製品種別を表すために用いられています．

ユビキタスIDでは，モノの一つ一つにIDを割り当てる関係上，JANコードをそのまま流用するのではなく，JANコードにさらに個別の識別子を割り当てることにします．そこで，図5のようにクラスBのID空間を割り当て，JANコードと個体識別子を割り当てることでJANコードを包含したコードを割り当てることが可能となります．このように，それぞれの必要性に応じてコード割り当てを行っていきます．

ただ，このようなコード割り当てをだれでもが行ってしまうと，どのIDを割り当てることができるのかがわからなくなってしまいます．そこで，このようなIDの割り当ての管理を行うための管理機関が必要になります．そのような機関としてユビキタスIDセンターがあります．

3 ユビキタスIDセンター

ユビキタスIDを使って，人間をとりまくコンピュータが実世界のモノや人を自動認識するためのしくみを提供し，ユビキタス・コンピューティング環境を実現するための機関としてユビキタスIDセンターが2003年3月に設立されました．

現在ユビキタスIDセンターは，ユビキタス・コンピューティングのプラットホームであるT-Engineの研究開発，普及を行っているT-Engineフォーラムによって運営されています．おもな活動内容としては，次のとおりです．

1) ユビキタスID空間の割り当て
2) ユビキタスID解決サーバの運用
3) ユビキタスID技術の研究開発
4) ユビキタスID技術の運用，実験
5) eTRONのための認証局

1)のユビキタスID空間の割り当てに関しては，先に述べたように，既存のID体系を取り込んだID空間をそれぞれのクラスに応じて割り当てています．

以降では，2)，3)に関して詳しく述べていきます．4)に関しては，最後に実験の紹介として食品トレーサビリティ実験の説

明を行います．

● ユビキタスID解決サーバ

ユビキタスID解決サーバは，前の項目で述べたユビキタスIDタグのIDをキーとしてサービスを提供するシステム・アドレスを検索する，分散型の軽量ディレクトリ・サービス・システムです．分散管理を前提としているユビキタスID解決サーバは，単一組織が管理するわけではありません．ユビキタスIDセンタでは，ルート・サーバとユーザから委託を受けたサーバのみを管理し，その他のサーバはサービス・プロバイダのような会社が管理することを想定しています．

また，日本のみならず韓国，中国，シンガポールなどさまざまな国々でも個別に管理する体制が整えられつつあります．

● ユビキタスID技術の研究開発の紹介

ユビキタスIDセンターでは，ユビキタスID技術を実現するうえでさまざまな研究活動を行っています．その中でも重視している研究として，セキュリティやプライバシの問題があります．ユビキタス・コンピューティング環境においては，次のような問題があります．

▶ 盗聴による問題

ユビキタスIDシステムにおける通信が傍受されることで，さまざまなプライバシ情報や秘密情報が漏洩するおそれがあります．たとえば，薬剤にタグがつけられた場合には，通信を盗聴することで，だれがどの薬を飲んでいるかという情報を推測できるかもしれません（図6）．

▶ RFタグ・データ読み取りによる情報漏洩の問題

RFタグ内の情報を無線通信により遠隔で読みとれてしまうと，そこから情報漏洩が発生する可能性があります．たとえば，ucodeタグのついた洋服を着ていた場合，そのucodeタグのIDを読み出すことで，いつどこでいくらで購入したかがわかってしまうことになりかねません（図7）．

▶ RFタグ・データ読み取りによる個人同定の問題

RFの持つIDを観察することで，モノの動きを把握できる場合があります．とくに，特定の個人だけが持つモノの場合，本人が気づかれないところで行動の追跡が行われている可能性があります（図8）．

これらの問題を解決するためにT-Engineフォーラム/ユビキタスIDセンターでは，eTRON（Entity and Economy TRON）や同定防止プロトコルなどの研究開発を行っています．eTRONは，コンピュータ化された社会活動において中核となる「情報」を安全に格納し，ディジタル情報基盤上で流通させることを可能とする，耐タンパ性を備えたハードウェアを用いた分散広域システム・アーキテクチャです（図9）．耐タンパ性を備えたハードウェア内で暗号化の鍵管理やデータ管理を行うことで，外部からデータ改ざんなどの不正なアクセスをすることができなくなるほか，ネットワーク経由でデータ通信を行う場合に，暗号化されたデータで通信されるために内容を解析することができなくなります．



図6　盗聴による問題



図7　RFタグ・データ読み取りによる情報漏洩の問題



図8　RFタグ・データ読み取りによる個人同定の問題

また，ucode解決サーバへのアクセスを行う場合にeTRONを用いることでアクセスする機器を特定することができるため，不正なアクセスを未然に防ぐことも可能となります．そして，RFタグのデータ読み取り時に個人の行動が追跡できないようなタグ・プロトコルの研究開発も行っています．

さらに研究開発例の一つとしてユビキタス・コミュニケータ（UC）があります（図10）．開発はYRPユビキタスネットワーキング研究所が主体となって行っており，UCを実際に使用した実証実験も行っています．

UCは，ユビキタス・ネットワーキング環境においてコミュニケーションを行うための汎用端末です．コミュニケーションには，モノとのコミュニケーション，環境とのコミュニケー

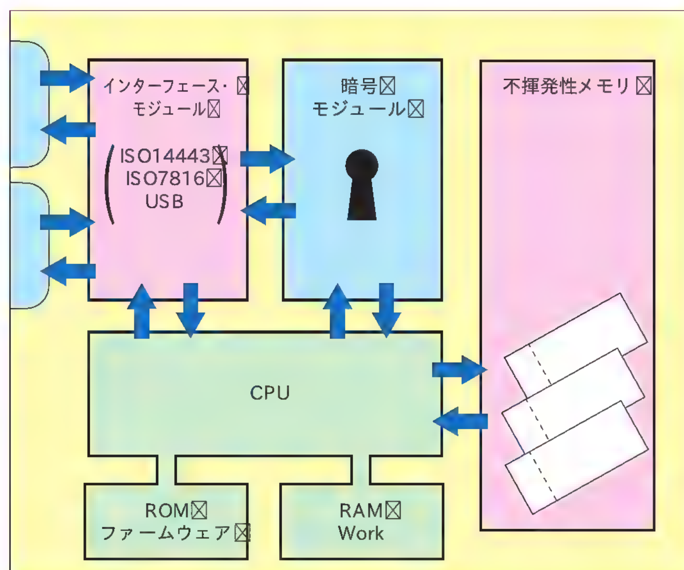


図9　eTRONアーキテクチャ

表2　ユビキタス・コミュニケータの仕様

ディスプレイ	VGA(480×640)，タッチ・パネル機能
内蔵通信モジュール	無線LAN，Bluetooth
非接触 インターフェース	13.56 MHz & 2.45 GHz デュアル・アンテナ
カード・ インターフェース	SD×1(SDIO対応) miniSD×1 SIM×1(eTRON用)
カメラ	正面×1(30万ピクセル) 背面×1(200万ピクセル，LED照明つき)
キー/スイッチ	カーソル・キー×1，正面キー×4 側面キー×1，電源スイッチ，底面スイッチ
赤外線	入力×1，出力×1
音声	ステレオ・スピーカ，マイク
生体認証	スイープ型指紋認証
外部 インターフェース	ステレオ・ヘッドホン出力， クレードル接続用コネクタ
クレードル・ コネクタ	シリアル，USBホスト，USBファンクション， マイク入力，ビデオ・デコーダ，外部電源， クレードル接続検出

ション，そして，人とのコミュニケーションがあります．

モノとのコミュニケーションとは，身の周りのあらゆるモノの情報をUCから得ることを意味しています．また,「環境とのコミュニケーション」とは，周りにあるネットワーク環境とのコミュニケーションを意味します．無線LANやBluetoothなどの無線インフラを介して周囲とデータのやり取りを行うことを意味します．人とのコミュニケーションとは，文字どおり人と会話を行うことを意味しています．たとえば，VoIP(Voice over IP)などの機能がそれに相当します．現在のUCの仕様を表2に示します．UCはすでに数回にわたって改良が行われ，改良のたびに新たな機能を追加してきました．マルチバンドのタグ・リーダ，無線LAN，Bluetoothなどの通信機能，MPEG，



図10　ユビキタス・コミュニケータ

JPEGコプロセッサなどを搭載しています．

4 ユビキタスIDセンターのタグの分類と認定タグ

今まではユビキタスIDの論理的な側面の話を中心にしてきましたが，ここでは物理的な側面の話をしたいと思います．

IDを格納するためのタグは，さまざまな方式が考えられます．近年注目を浴びている技術であるRFタグや，従来多く用いられてきたバーコードなども識別を行うための技術の一つです．また，RFIDとひと言でいってもさまざまな方式があり，用いている周波数帯，通信プロトコルがそれぞれ異なっています．このようなタグの分野では，デファクト・スタンダードのようなある一つのタグだけをサポートすれば十分であるということは考えにくいものです．なぜならば，コスト面だけをみればバーコードのほうがRFIDよりも安価ですし，機能面から考えた場合でも，RFタグで用いる周波数帯が違えば通信距離や性能が変わってきます．また，貼り付ける対象となるモノの大きさや材質なども考慮する必要があります．水分がある場合には2.45GHzのRFタグは不向きであるとか，小さいモノにはバーコードを印字するのが難しいなど，考慮すべき点はさまざまです．また，タグには多くの特許が存在するため，一方式に限定してしまうのは非現実的です．

これらのことを考えると，ある規格を一つ選択するというのではなく，さまざまな方式に対応していくのが正しい方法だといえます．

● ユビキタスIDセンターの認定タグ・クラス

そこで，ユビキタスIDセンターではある認定基準を設けて，申請があったタグに対して認定を行っています．基本的には，UCで通信可能であるタグであれば標準タグとして認定する方針です．ただし，セキュリティ・レベルに応じたクラスの分類と物理インターフェースに応じた分類をしています．表3にクラスの分類を，表4にはインターフェース・カテゴリを示しま

表3　セキュリティ・レベルに応じたクラスの分類

クラス	提供するセキュリティ機能
クラス0	データ欠損検出機能
クラス1	耐物理的複製/耐物理的偽造
クラス2	同定防止機能
クラス3	耐タンパ性(物理的，論理的) 資源別アクセス制御管理機能
クラス4	未知ノードとの安全な通信
クラス5	時刻に依存した資源管理機能
クラス6	内部プログラム/セキュリティ情報の更新機能

表4　インターフェース・カテゴリ

	カテゴリ0 印刷タグ	カテゴリ1 RFタグ	カテゴリ2 アクティブRFタグ	カテゴリ3 アクティブ赤外タグ
クラス0 データ欠損検出機能	○	○	○	○
クラス1 耐物理的複製 耐物理的偽造	－	○	○	○
クラス2 同定防止機能	－	○	○	○
クラス3 耐タンパ性(物理的，論理的) 資源別アクセス制御管理機能	－	○	○	○
クラス4 未知ノードとの安全な通信	－	○	○	○
クラス5 時刻に依存した資源管理機能	－	－	○	○
クラス6 内部プログラム/ セキュリティ情報の更新機能	－	－	○	○

○：原理的に存在する，－：原理的に存在しない

す．セキュリティ・レベルに応じたクラス分けは，耐タンパ性や同定防止機能などさまざまなセキュリティに対する機能を含んでいるかどうかで分類されます．また，インターフェース・カテゴリは，UCが備えるタグ・インターフェース装置に対応した分類となっています．

● セキュリティ・クラス

以降では，セキュリティ・クラスに関してもう少し詳しく説明します．セキュリティ・クラスに関しては，現在T-Engineフォーラム内のワーキング・グループである「ucodeタグ技術WG」で議論されている内容です．今後変更される可能性はありますが，現在の指針を示しておきます．

▶ クラス0

通信データ欠損検出機構を備えたucodeタグをクラス0としています．通信データ欠損とは，タグとリーダとの通信を行うときに外乱によってデータの一部が破損する，もしくは光学タグの物理的な欠損をいいます．そのような欠損が起こった場合でも検出することができるようなしくみを，通信データ欠損検出機構と呼びます．

▶ クラス1

次に，クラス0の機能に加え耐物理複製・偽造機能を備えたタグをクラス1のタグとしています．耐物理複製・偽造とは，物理的に同一もしくは類似のものを作成することが困難なことを指します．ここでの困難さとは，一般の人々が持っている設備を用いて複製，偽造を行う場合の困難さを意味しています．

▶ クラス2

クラス1のタグの機能に加えて，同定防止機構を備えているタグをクラス2のタグとしています．同定防止機構とは，先に述べたように通信状況や通信内容，通信方法を特定されないようにする機構を指しています．

▶ クラス3

クラス2の機能に加えて，耐タンパ機能，資源別アクセス制御管理機能を備えたタグをクラス3としています．耐タンパ機能とは，タグに格納されている情報を不正に読み出せないようにする機能のことです．耐タンパ性には物理的耐タンパ性と論理的耐タンパ性の2種類があります．物理的耐タンパ性は，物理的な解析，たとえばメモリ中の電気信号を物理的に読み出すなどによって，データを読み出すことができないようにする性質のことです．また，論理的耐タンパ性は，ある特定の論理的な処理を行うことで不正にアクセスすることができるようなしくみがないという性質のことです．また，資源別アクセス制御機能とは，資源アクセス者の権限クラスに応じた格納資源ごとのアクセス制御を行う機構です．

▶ クラス4

クラス4の機能に加えて，未知のノードとの安全な通信路構築機能を備えているタグをクラス4のタグとしています．タグのデータをネットワークを介してやり取りするときに，事前に秘密鍵を共有していない不特定ノードに対しても安全なデータ通信路を確立可能とする機能です．

▶ クラス5

さらにクラス4の機能に加えて，内部セキュア・クロックを用いた資源管理機能を備えたタグをクラス5のタグとしています．内部セキュア・クロックを用いた資源管理機能とは，キャリア・データやセキュリティ情報，タグ機能動作の次元管理機能のことです．たとえば，データの有効期限を設定したり，ある一定時間が経過すると動作を停止させたりする機能があげられます．

▶ クラス6

最後に，クラス5の機能に加えて，内部プログラム・セキュリティ情報の更新機能を備えたタグをクラス6のタグとしています．内部プログラム・セキュリティ情報の更新機能とは，ファームウェアの更新やセキュリティ・パッチの適用など，使用状況に合わせた最適なセキュリティ機能を維持することが可能な保守機能のことを指します．

● 現在ユビキタスIDセンターで認定されているタグ

次に，現在認定されているタグについて紹介します．現在認定されているのは，

● バーコード，二次元コード(サトー，凸版印刷，大日本印刷，



図11　二次元コード

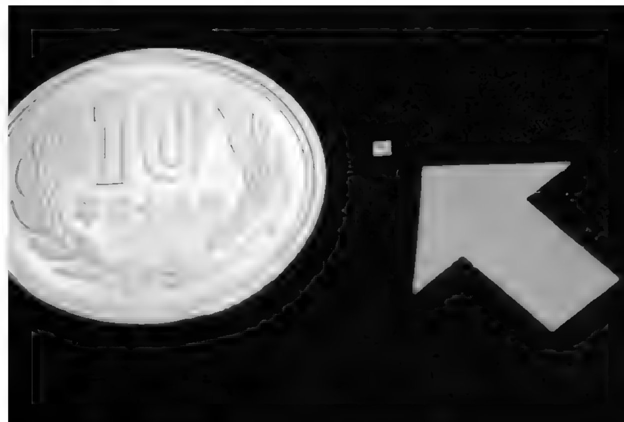


図12　無線自動認識ICミューチップ（クラス1）

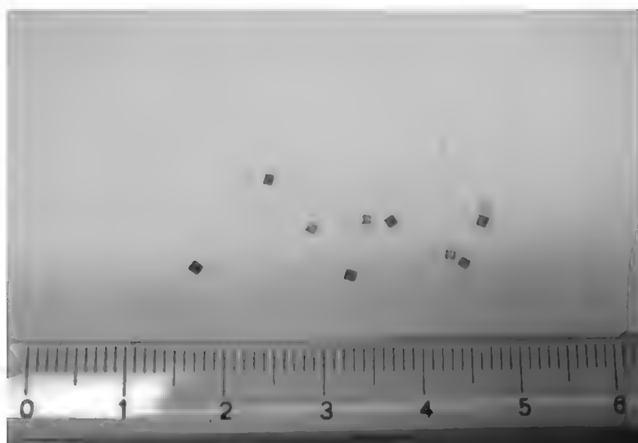


図13　RFタグT-Junction（クラス1）



図14　FRAMを搭載した大容量高速ICタグ用LSI MB89R116，MB89R118



図15　ユビキタスIDチップeTRON/16（クラス4）

図11）
- ミューチップ（日立製作所，図12）
- T-Junction（凸版印刷，図13）
- MB89R116，MB89R118（富士通，図14）
- eTRON/16（YRP UNL，東大，ルネサステクノロジほか，

図15）

があります．バーコードに関しては，バーコードそのものではなくユビキタスIDセンターで割り当てているコードを出力することができる印刷機に対して認定しています．また，次の項で述べる食品トレーサビリティの実証実験においても，ユビキタスIDセンターの認定タグが用いられています．

5 ユビキタスID技術の応用例：食品トレーサビリティ

● 食品トレーサビリティの概要

YRPユビキタス・ネットワーキング研究所では，2003年度後半より，ユビキタスID技術の応用拡大の一環として食品トレーサビリティ・システムの開発および実証実験を実施しました．

おりしもBSEや食肉偽装，輸入食品の残留農薬などの問題が社会に大きくクローズアップされていた時期であり，あらゆるモノにコンピュータを付加し，人間生活を豊かにサポートするユビキタス・コンピューティング社会の応用モデルとして，

New Products――日立と千住金属工業，「高温無鉛はんだペースト」を開発
（株）日立製作所生産技術研究所と千住金属工業（株）は，銅粉と錫系粉の複合材料からなる「高温無鉛はんだペースト」を共同で開発した．また，「2004実装プロセステクノロジー展」にて展示を行った．

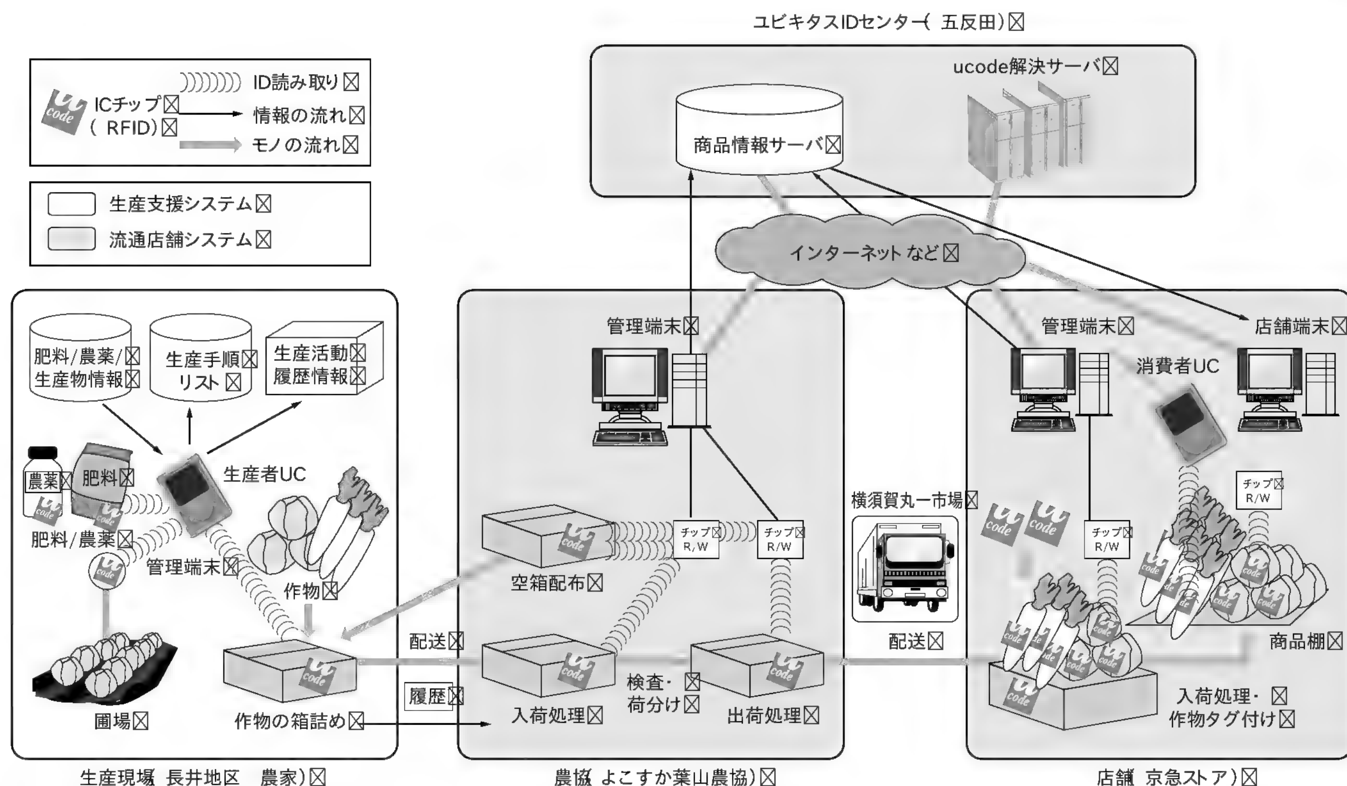


図16 食品トレーサビリティ・システムの構成

食品の安心・安全を消費者にもたらす食品トレーサビリティは，まさにタイムリな取り組み分野となりました．

食品トレーサビリティは，食品の生産履歴や流通履歴を正確に把握し，その情報を消費者が追跡するトレーシング(tracing)と，逆に事故の発生後に，生産者や流通・販売者が，原因と被害範囲をすばやく特定するトラッキング(tracking)の二つの機能を実現します．また，トレーサビリティ・システムでは，RFIDやバーコードなどのタグを，トレースの対象となる食品に貼付し，生産履歴や流通履歴などの情報を，タグ内のucodeが示すアドレスに存在する商品情報サーバ・データベースに格納します．これにより最終的に，消費者が食品を入手または使用する際に，タグによって関連付けられた情報(生産履歴や流通履歴)を確認することができるということになります．

YRPユビキタス・ネットワーキング研究所は，システムの構築にあたって，RFIDやユビキタス・コミュニケータといったユビキタスID技術の中核をなす先進技術要素を積極的に取り入れ，その有効性検証に役立てることとしました．

● 食品トレーサビリティ・システムのシステム構成

今回，検証を行った食品トレーサビリティ・システムの全体の構成を図16に示します．

▶ ユビキタスIDセンター

ユビキタスIDセンターには，ucode解決サーバを設置します．ucode解決サーバは，生産現場の農薬・肥料や食品を格納する箱，そしてダイコンやキャベツなどの食品そのものに貼り付けるRFIDタイプのucodeタグに格納されたucodeを元に，それらの品物の情報を格納した商品サーバのアドレスを解決して通知するucode解決サービスを提供します．ucode解決に用いられる通信プロトコルは，暗号認証通信基盤であるeTP(Entity and economy Transfer Protocol)をベースとしたucodeRP(ucode Resolution Protocol)で，ここでもユビキタスID技術をいち早く試験導入し，実証することを可能としました．

▶ 生産支援システム

生産支援システムでは，日々の農作業において生産者に配布した生産者ユビキタス・コミュニケータを使用し，農薬・肥料の散布といった生産活動履歴を記録するとともに，農薬使用時期や使用制限回数などの基準をチェックし，単なる履歴の記録に止まらず農家の生産活動を支援するしくみを提供します．

この生産支援システムを実現するにあたって，農薬や肥料のパッケージにはucodeタグ(クラス3，下位スマート・タグ・タイプ)を貼付します．タグの貼り付けによってユビキタス・コミュニケータがこれらのモノを認識できるようになり，先に挙げたきめ細かな生産支援機能が実現可能となるわけです．

▶ 流通・店舗システム

農協などの集荷場から店舗への流通段階では，各管理端末を使用して，食品の流通履歴を記録し，商品情報サーバへ追記していきます．食品の収まっている箱にはucodeタグ(クラス1,



図17　KIOSK端末

下位RFタグ・タイプ）が貼り付けられ，これらのタグを管理端末が読み込み流通履歴を記録します．

また，店舗に入荷した食品は箱から取り出され，食品一つ一つにucodeタグ（クラス1）が貼り付けられます．このとき，店舗の管理端末で箱のucodeタグを読み取り，食品に貼り付けるucodeタグと箱との関連情報を紐付けていく作業が行われます．

▶ 店舗・消費者表示システム

トレーサビリティ・システムの最終段階では，消費者が店頭や家庭などで食品を手にとり，貼り付けられたucodeタグを図17のような店舗端末や消費者用ユビキタス・コミュニケータで読み取らせることにより，商品情報を閲覧することができます．

ucodeタグに記録されたucodeを，店舗端末やユビキタス・コミュニケータがucode解決サーバに解決依頼し，結果を参照し商品情報サーバのアドレスにアクセスします．

これらの端末はucodeRPを搭載しているほか，RFIDの読み取り装置も付属するなど，ユビキタスID技術を豊富に使用して応用システムを実現しています．とくにコンパクトで携帯性に優れ，豊富なインターフェースを搭載したユビキタス・コミュニケータはT-Engineをベースに開発されており，今後ほかの応用システムにもリファレンス・システムとして役立てることができるでしょう．

● 実証実験への取り組み

YRPユビキタス・ネットワーキング研究所は，T-Engineフォーラムが農林水産省より受託した「平成15年度食品トレーサビリティ開発事業」に参加し，2003年11月から2004年2月にかけて，よこすか葉山農協，京急ストアと共同で食品トレーサビリティ実証実験を実施しました．

ダイコン農家4軒，キャベツ農家4軒の計8軒のよこすか葉山農協組合員に実験用圃場を提供していただき，合計2万5千個の農作物を実験用に栽培しました．

これらの作物は2004年1月には東京・神奈川の京急ストア3店舗に配送され，消費者が店頭でucodeタグを店舗のKIOSK端末で読み取らせ，商品の生産履歴や農家の顔写真，作物育成に対するこだわりといった情報を公開しました．幸い冷夏の影響も少なく作物も順調に育ち，予定どおりの数量を約1か月間で出荷し，店頭にて販売しました．

実証実験を実施した結果，さまざまなことがわかってきました．全体的な効果としては，通常のトレーサビリティ・システムの基本機能であるトレース情報の提供に加え，「生産者支援とトレース情報記録の統合」および「食品トレーサビリティ・システムへのユビキタスID技術の適用」について有効性を検証でき，高い評価をいただきました．

● 今後の予定

YRPユビキタス・ネットワーキング研究所は，2003年度の実証内容を踏まえ，2004年度も継続して食品トレーサビリティシステムの開発・実証を進めていきます．昨年度構築したシステムはトレーサビリティの基本機能であるトレーシング・トラッキングに対応し，トレーサビリティ・システムの必要条件は満たすことができており，本年度はこのシステムに追加機能を盛り込み，より使い勝手の良いしくみにしあげることをめざしています．

まず，扱い品目の拡大を目標とします．昨年度の扱い品目は青果物（ダイコン，キャベツ）のみで，必ずしも品目としては多いものではありませんでした．今年度は対象を食肉，加工品などの生産・流通形態にも広げ，より広範囲に食品トレーサビリティ事業をカバーできるようにします．

また，早期の実用化を視野に入れ，最先端のRFIDやユビキタス・コミュニケータだけでなく，既存システムで活用されているバーコードにも対応し，低コスト化を図る予定です．この際，使用するバーコードはucodeタグとして認定されたものが採用されます．

これらの改良により，実証実験だけでなくより実用システムに近づいたパッケージ・システムとしてしあげていく予定です．

また，T-Engineフォーラムが昨年度に引き続き取り組む農林水産省「平成16年度食品トレーサビリティ開発事業」にも参加します．本事業では，果物，豚肉（加工品含む）などのトレーサビリティ情報を三越，京急ストア店頭で提供する実証実験を実施する予定です．

おだて・ひでき/こばやし・しんすけ
YRPユビキタス・ネットワーキング研究所



小型基板とインターネットで実現する

H8S マイコンによる RFID システムの作成

松永 隆文



RFIDリーダ/ライタを用いたシステムというと，とかく大規模なものを想定しがちだが，現在では小型マイコン・ボードとRFIDリーダ/ライタを接続することにより，手軽に開発が行える．
本章では，H8S マイコン・ボードにLinuxライクな独自OSを搭載した「eBoss-1」と，RFIDリーダ/ライタをRS-232-Cで接続し，相互に通信することによりRFタグの読み込みを行う．サンプル・プログラムはC言語で書かれているほか，一般的なPOSIX APIが使われている．（編集部）

1 はじめに

今,“ユビキタス”ということばが巷(ちまた)にあふれています．一般的にユビキタス・コンピューティングとは，目に見える形でコンピュータが存在せず,「生活環境の中にコンピュータ・チップとネットワークが組み込まれ，ユーザは場所や存在などを意識することなく利用できるコンピューティング環境」と定義されているようです．これは，場所や時間，そして環境などの制約を受けずに,情報の伝達が可能になることを意味します．

しかしながら,「ユビキタス」ということばから描くイメージが先行し，数多くの技術的・産業的・社会的にクリアしなければならないハードルが待ちかまえています．とくに，RFタグは，ユビキタス・コンピューティングには欠かせない要素技術であり，ユビキタス社会実現の過程の中で実証実験が行われています．

RFタグを物品につけることにより，

- 荷物の出発地点/経由地点/到着地点など，物理的な移動のリアルタイム管理が可能（図1）
- 物品のありかを即座に入手できるため，ロケーション管理が不要
- 事務処理の効率を向上
- 暗号機能を備えたRFタグをカギとして利用することでセキュリティ・レベルを向上

などが実現できます．さらに応用例としては，

- 自動車のキーに内蔵したRFタグとエンジン・コントローラの認識コードが一致したときのみに，エンジンを始動するシステム．これはすでに実用化されている．さらに紙幣，小切手，株券，チケットなどにRFタグを埋め込めば，偽造や不正流通の防止が容易になる
- 道路や街灯，信号機，横断歩道など特定の場所にRFタグを埋め込むことにより，そこから地域の情報が得られ，地図に頼らず容易に目的の場所に到達することができる．これは高齢者や視覚障害者，車椅子利用者などがスムーズに移動できるバリア・フリー社会の構築ができる

などがあります．これらが普及するためには「コスト的な視点」と「わかりやすく簡単な適応事例」を世の中にアピールすることが重要と考えられます．

2 RFタグの現状

RFタグの現状を表1に示します．これら各方式の中では，電磁誘導方式（13.56Mz）とマイクロ波方式（2.45GHz）の普及が進んでいます．まだ国内では未認可ではあるものの，UHF帯は遠距離通信が可能なため，車両，コンテナなどの物品検品用途での普及が進むと考えられています．

3 RFID端末（リーダ/ライタ）

RFIDリーダ/ライタの端末形態を表2に示します．PCやPOS端末経由でのデータ収集では，システム全体のコストや

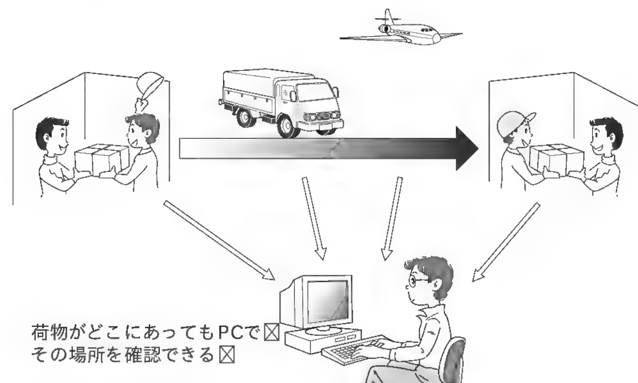


図1 荷物の物理的な移動の管理

表1　RFタグの現状

方　式	通信距離	周波数	特　徴
静電結合 方式	数mm	—	静電気による誘導作用， 電気ノイズには強い
電磁結合 方式	～10cm	300～500kHz	相互誘導作用 汚れ，水，油などには 強い
電磁誘導 方式	～30cm	60～135kHzまたは 13.56MHz （ISO15963, ISO14443）	電磁誘導作用 無線タグの形は自由に 近い
マイクロ波 方式	～100cm	2.45GHz	超短波 （マイクロ波を利用） 通信距離を長くできる
UHF帯 方式	～6m	915MHz近辺 （国内では未認可）	UHF帯を使用 アンテナにくふうが必要 （タグの方向に依存しない） （200枚/sの読み取りが可能．米国マトリックス社）

表2　RFID端末形態

端末形態	特　徴
定置リーダ， ライタ	各種端末やFA現場，作業場の据え置き型での固定位置に設置される端末．電源，重量，形状に左右されず，高性能なリード/ライトが可能．ただし，小形化の要求が高い
ハンディ端末	機器を携帯してデータの読み取りと書き込みを行う．作業指示の受信や作業結果データのホスト機宛て通信にSS無線形式の通信手段を内蔵するものもある．バーコード・スキャナなどを内蔵するタイプもある．商品/製品の検品，棚卸などで利用される．携帯電話への応用も考えられている
車載端末	作業用フォークリフト，構内運搬車両，クレーン操縦席などに固定設置し，データの読み取りと書き込みを行う．ホスト宛て送受信については，ハンディ端末と同じ．重量制限がないため，高性能な物が多い．最近は，産業廃棄物管理用途がある

注：上記端末形態の中で，“定置リーダ，ライタ”用途での利用は一般的にRFタグ/リーダ・メーカの各社ともPOS端末やPCなどへの接続を想定し，単独での利用に関してあまり考慮されていないのが実状である．

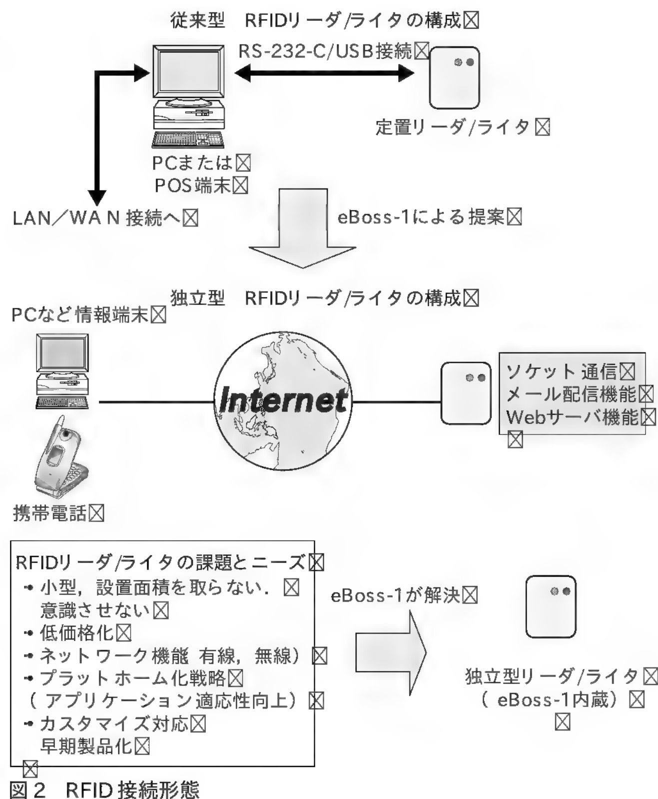


図2　RFID接続形態

トータルな機器コストが上がるだけでなく，メンテナンス性の低下をも招き，市場への普及を妨げる要因となりえます．また，設置される場所などの制限により，上記システムは実用的でないという現状もあります．

したがって，“定置リーダ/ライタ”がRFタグのスキャナ単体機能だけでなく，ユビキタス社会には必須のインターネット接続機能をもたせ，なおかつ，メンテナンス・フリーで小型な「独立型リーダ/ライタ」に対する要求が高まっています（図2）．また，RFタグの情報は，利用されるアプリケーションごとに異なるため，情報を処理するリーダ/ライタのプラットホーム化が重要な鍵となります．製品の早期立ち上げ，カスタマイズ対応は今後のリーダ/ライタの必須条件となります．

このような背景から，ユビキタス社会に適応した，容易でシステム全体としてコストダウンが見込めるソリューションが求められています．

eBoss-1（Embedded Basic Operating Smart System）は，「POSIX準拠 超小型リアルタイム・カーネル」+「H8Sマイコン搭載ボード」で構成されています．カーネルは，超小型サイズでH8Sマイコンの内蔵フラッシュ・メモリでの動作を可能とし，Linuxと同等の環境（Linuxライクな環境）を実現しており，RFID市場のニーズに合致したものを考えて設計しています．



eBoss-1 + H8Sボードによるリーダ/ライタ制御とLAN制御

● eBoss-1の特徴

カーネルのおもな特徴を表3に，ボードの外観を写真1に，ボード仕様を表4に示します．

● RFIDリーダ/ライタ評価ユニット

eBoss-1を使用したLAN対応RFIDリーダ/ライタ評価ユニット（写真2）を用いて説明します．

RFIDリーダ/ライタ（写真3）は，マルチリード/ライト（Mifare, i-Codel，ISO15693タグ）が可能なものを使用しています．

RFIDリーダ/ライタのおもな機能は以下のとおりです．

- RFIDリーダ制御（RS-232-C制御）
- LAN（TCP/IP）
- CFによるPHS，ATA（VFATサポート）
- LCDに状態表示
- RFタグ読み取り時にLEDやブザーを点灯（DIO）
- RFタグ情報により以下の機能が利用できる

Information ――コンピュータ・ウィルス，トロイの木馬型が継続して増加中
マカフィー（株）のコンピュータ・ウイルスの検知状況によると，9月にもっとも多く検知されたウィルスはトロイの木馬型である「Exploit-MhtRedir.gen」だった．さらに，検知数のトップ4のすべてがトロイの木馬型だった．

表3　eBoss-1カーネル概略仕様

1) Linuxライクな環境を実現 POSIX準拠インターフェースを採用 GPLリソースの有効活用が可能 コマンド・シェル機能を実装 GCCベースでの開発環境
2) ROM/RAMともに256Kバイトで動作可能な超小型カーネル
3) TCP/IP，DHCP，DNSレゾルバ，pppd，telnetd，ftpd，httpd，SMTPを標準実装 TCP/IPはkeep alive，httpdはCGI機能付き
4) 数百msの超高速起動
5) 外部プログラム起動可能（ダイナミック・リンク起動）
6) CompactFlashインターフェース・ドライバを標準実装 P-in，@Freed（無線LAN，AirH"はオプション） メモリ（ATA）カードはVFATをサポート
7) ハード・リアルタイムをサポート． POSIX準拠pthreadインターフェースに対応



写真1　eBoss-1ボードの外観（CFカードは製品に含まれない）

① タグ情報のLANによるソケット通信
② タグ情報のメール送信（SMTP，WAN接続可能．P-inなどと接続）
③ CF（ATAカード．VFAT）へのデータの保存

● RFIDリーダ制御について

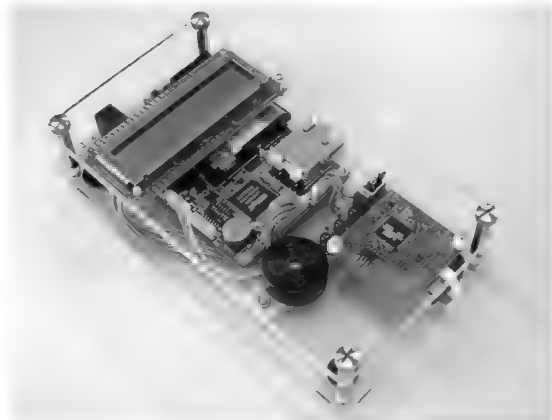
RFIDリーダとeBoss-1の接続例を図4に示します．

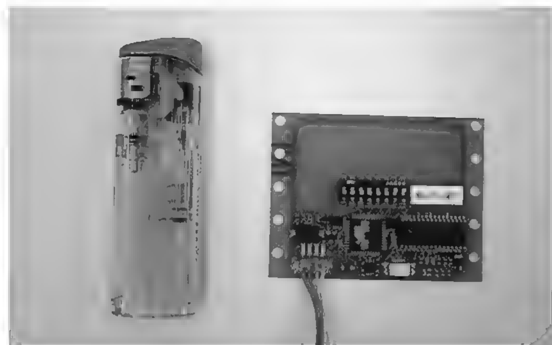
各社のRFIDリーダのリード/ライト制御コマンド仕様は，秘密保持契約締結が原則となっています．したがって，本コマンドを公開することはできませんが，RS-232-C制御をPOSIX準拠APIで行えます．RFタグ内の情報の扱いについては，アプリケーションごとに異なります．

eBoss-1は，POSIX準拠カーネルであることから，UNIX系，Linux系アプリケーションの開発経験があれば，同一のAPIで制御が可能です．また，アプリケーションの開発はGCCを使用して行います．また，Windows環境でGCCを使った開発をサポートするCygwinも開発キットに同梱されています．

表4　ボード仕様

項　目	内　容		
CPU	H8S/2339EF（F）25MHz，16ビット		
メモリ	RAM	MPU内蔵SRAM	：32Kバイト
		外部SRAM	：1Mバイト（拡張可能）
	ROM	MPU内蔵FROM	：384Kバイト
		外部FROM	：2Mバイト
	EEPROM：8Kビット		
デバイス	Ethernet，RS-232-C×3チャネル，CF，RTC，A-Dコンバータ，DI/DO 4チャネル，I^{2}C，LCD，H8S拡張バス		
寸法	90×60mm（ほぼ名刺サイズ）		


写真2　RFIDリーダ/ライタ評価ユニットの外観


写真3　RFIDリーダ/ライタの外観

1
2
App
3
4
5
6

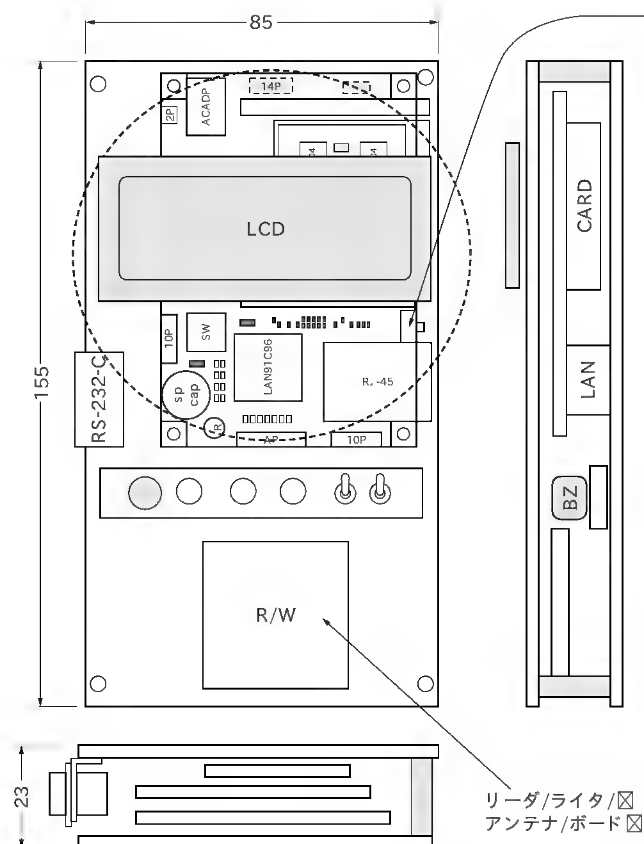


RFタグから読み取ったデータをネットワーク経由でサーバに送信する例

● Linuxにおけるアプリケーション開発

eBoss-1はPOSIX準拠カーネルとなっているので，Linuxでのアプリケーション開発の経験と，RS-232-C制御の経験があれば問題なくアプリケーション開発が可能です．実際のプログラム・コードはRed Hat Linux 7.0/8.0で動作するアプリケーションとまったく同一のプログラムで動作します．

そこで，ここではRFタグから読み取ったデータをネット



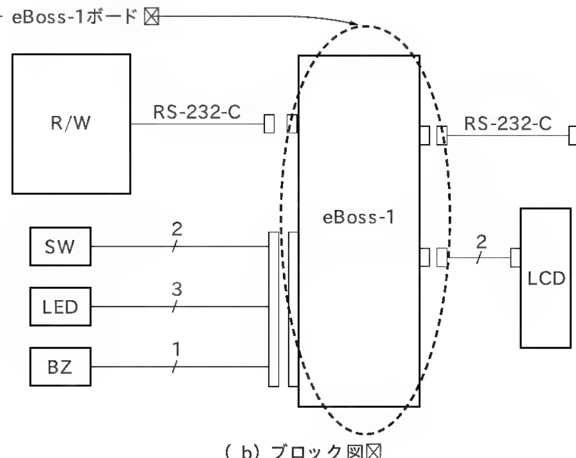


図3　RFIDリーダ/ライタの構成

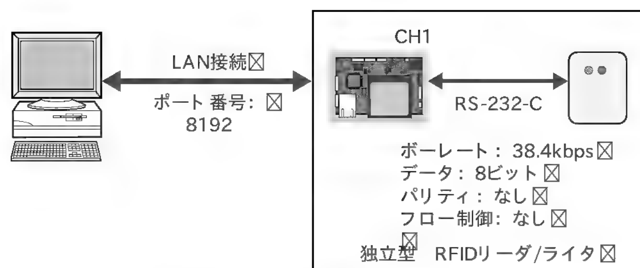


図4　RFIDリーダとeBoss-1の接続例

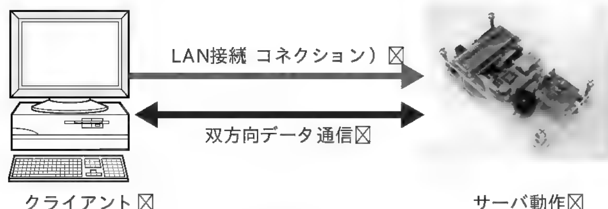


図5　ソケット・サーバ通信構成図

表5　ハードウェア・デバイス

デバイス・ドライバ	デバイス名	用途
キャラクタLCD	/dev/lcd	RFIDタグ種別，データ表示用途
ディジタル出力(DO)	/dev/do	LED制御，ブザー制御
COMデバイス(RS-232-C)	/dev/ttySC1	RFIDリーダ/ライタ制御
ネットワーク・デバイス(LAN)	/dev/eth0	RFIDタグ情報のソケット通信

表6　通信仕様

デバイス	詳細	設定
RS-232-C	使用デバイス	ttySC1(CH1)
	ボーレート	38.4kbps
	データ長	8ビット
	パリティ	なし
	フロー制御	なし
LAN	ポート番号	8192

ワーク経由でサーバへ送信するプログラムを例にして，実際のプログラムの詳細について説明します．これは，RFIDリーダから読み取ったデータを，単純なソケット通信によるデータ送信を行うという，ソケット・サーバ型のプログラム例になります(図5)．

ソケット・サーバ型は，RFIDリーダ/ライタがコネクション接続した相手に対してソケット通信でRFタグ情報を送信します．

本アプリケーションで使用するデバイスを表5に示します．ここにあるデバイス名を使えば，標準的なドライバ・インターフェースでアクセスできます．そのほかにeBoss-1カーネルではRAMディスク，シリアルEEPROMをファイル・システムとして扱えます．またCF(ATAカード，P-inカード)のデバイス・ドライバが標準でサポートされています．

● ヘッダ部

リスト1がソケット・サーバ・タイプのプログラムのヘッダ部です．ネットワーク接続(LAN)，RS-232-Cの通信条件を表6に示します．RS-232-Cの通信条件はRFIDリーダ仕様に依存します．また，ここではポート番号として8192を使用していますが，knownポート番号以外を指定することから，

リスト 1　socksv.c(ソケット・サーバ・タイプ)のヘッダ部

```
/*
 * Copyright (c) 2004 Computer Hi-Tech, Inc.
 *
 * RF-ID  R/W  Controll  sample
 * Type =ソケット通信サーバのサンプル
 *
 */

#include    <ctype.h>
#include    <fcntl.h>
#include    <netinet/in.h>
#include    <netdb.h>
#include    <sys/select.h>
#include    <termios.h>
#include    <unistd.h>
#include    <errno.h>

#include    <knet.h>

#include    <wchar.h>
#include    <stdlib.h>
#include    <string.h>

/* RS-232-C ch1 port RFID Control */
#define TGT_DEVICE  "/dev/ttySC1"

#define DEFAULT_PORT     8192        /* default port number */
#define DEFAULT_C_TIME   0
#define DEFAULT_CHARS    0
#define DEFAULT_BPS      B38400      /* 38400bps    */
#define DEFAULT_CSIZE    CS8         /* 8bit        */
#define DEFAULT_PARITY   0           /* non parity */
#define DEFAULT_STOP     0           /* 1bit        */
#define DEFAULT_FLOW     0           /* no flow     */
#define BUFSZ            1024
/* read data length NDA secret */
#define xx               RFID

typedef struct  {
    int      rsfd, lstn, conn, lcdfd;
    int      bps, csize, parity, stop, flow;
    int      portno;
    char    *dev;
    char    *ttybuf, *sckbuf;
    int      ttycnt, ttystp, ttyenp;
    int      sckcnt, sckstp, sckenp;
} scv_session_data_t;
typedef scv_session_data_t* scv_session_t;
```

リスト 2　各種デバイス処理部

```
/*
*  lcd_open     : キャラクタ LCD デバイスを開く
*/

static int lcd_open(void)
{
    in      sfd;
    if((sfd = open(g_lcd, O_WRONLY)) < 0) {
    fprintf(stderr, "can't open %s¥n", g_lcd);
        usage();
        return -1;
    }

    return sfd;
}

/*
*  do_open     : ディジタル出力(DO)デバイスを開く
*/
static int do_open(void)
{
    int     sfd;

    if((sfd = open("/dev/do", O_RDWR)) < 0) {
        fprintf(stderr, "can't open %s¥n", g_do);
        usage();
        return -1;
    }

    return sfd;
}
/*
 *  rs_open     : シリアル通信ポートを開く
 */
static int rs_open(scv_session_t cn)
{
    int             sfd;
    struct termios  newtio, oldtio;

    /* シリアル通信デバイスを開く */
    if((sfd = open(cn->dev, O_RDWR | O_NOCTTY)) < 0) {
        fprintf(stderr, "can't open %s¥n", cn->dev);
        return -1;
    }

    /* 現在のシリアル・ポートの設定を待避させる */
    if(tcgetattr(sfd, &oldtio) < 0) {
        perror("tcgetattr");
        close(sfd);
        return -1;
    }
    /* 新しいポート設定の構造体をクリア */
    memset(&newtio, 0, sizeof(newtio));

    /*
        CLOCAL  : ローカル接続，モデム制御線を無視する
        CREAD   : 受信文字(receiving characters)を有効にする.
        HUPCL   : 最後のプロセスがデバイスをクローズした後,
                  モデムの制御線を切断する
    */
    newtio.c_cflag = cn->bps | cn->csize | cn->parity
                     | cn->stop;
    newtio.c_cflag |= (CLOCAL | CREAD | HUPCL);

    if(cn->parity == 0)
        /* パリティのエラーは無視する */
        newtio.c_iflag = IGNPAR;
    else
        /* パリティ，フレーム・エラー発生前の文字に
           ¥377¥0 を付加 | 入力のパリティ・チェック有効 */
        newtio.c_iflag = PARMRK | INPCK;

    if(cn->flow == CRTSCTS)
        newtio.c_cflag |= cn->flow;
    else if(cn->flow != 0) {
        newtio.c_iflag |= cn->flow;
        newtio.c_cc[VSTART] = 0x11;
        newtio.c_cc[VSTOP] = 0x13;
    }

    /* Raw モードでの出力 */
    newtio.c_oflag = 0;
    /* すべてのエコーを無効にし，プログラムに対して */
    /*              シグナルは送らせない      */
    newtio.c_lflag = 0;

    /* すべての制御文字を初期化する */
    newtio.c_cc[VTIME] = DEFAULT_C_TIME;
    newtio.c_cc[VMIN]  = DEFAULT_CHARS;

    /* モデム・ラインをクリアし，ポートの設定を有効にする */
    if(tcsetattr(sfd, TCSANOW, &newtio) < 0) {
        perror("tcsetattr");
        /* ポートの設定をプログラム開始時のものに戻す */
        tcsetattr(sfd, TCSANOW, &oldtio);
        close(sfd);
        return -1;
    }

    /*** バッファ・カウンタの初期化 ***/
    cn->ttycnt = cn->ttystp = cn->ttyenp = 0;

    return sfd;
}
```

ネットワークの管理者と相談してポート番号を決めます．なお，LAN（Ethernet，/dev/eth0）は，カーネルが起動時にドライバがデバイスをオープンするので，open()処理などは必要ありません（Linuxカーネルと同じ）．

RS-232-Cの通信仕様は，RFIDリーダの設定に依存します．また，#includeするヘッダ・ファイルの構成は，Linuxとほとんど共通です．使用するライブラリ関数に応じて#include宣言します．

● 各種デバイス処理部

リスト2は各種デバイスの初期化処理部です．各デバイスをオープンし，必要ならばパラメータのセットを行います．

- lcd_open()は，キャラクタLCDデバイスをオープンする
- do_open()は，ディジタル出力デバイスをオープンし，LED制御やブザー制御の準備を行う

これらは，ターゲット・ボードのハードウェアに依存したデバイスですが，openとwriteもしくはioctlで簡単に制御できます．

- rs_open()は，シリアル・ポートをオープンして，通信パラメータを設定する．シリアル・ポートの通信パラメータの設定には，struct termios構造体に値をセットし，tcsetattr()関数で設定する．

● ネットワーク処理部

リスト3は，サーバ・ソケットの制御部です．

- sock_listen()は接続要求待ちのソケットを作成する
- sock_accept()はクライアントからの接続要求を受け付け，コネクションのソケット・ディスクリプタを返す．データの送受信は，これを通して行われる

次いでリスト4は，内部のデータ・バッファにあるデータをLANへと送信する処理を行います．send()関数は，実際に送信できた（TCPの送信バッファに格納できた）データのバイト数を返すので，その数だけデータ・バッファのカウンタを進めます．

● メイン処理部

リスト5はメインの処理部です．

リスト3　ネットワーク処理部

```
/*
 * sock_listen : ソケット接続待ち準備
 */
static int sock_listen(int *sockFd, int pno)
{
    struct sockaddr_in   svAddr;

    if((*sockFd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        fprintf(stderr, "Error:sock_listen -- socket");
        return -1;
    }

    memset(&svAddr, 0, sizeof(svAddr));
    svAddr.sin_family = AF_INET;
    /* ポート番号設定 */
    svAddr.sin_port = htons(pno);
    /* サーバ側のIPアドレスはシステム側の設定にゆだねる  */
    svAddr.sin_addr.s_addr = INADDR_ANY;

    /* ソケットのアドレスを指定する */
    if(bind(*sockFd, (struct sockaddr *)&svAddr,
             sizeof(svAddr)) < 0) {
        perror("Error:sock_listen -- bind");
        close(*sockFd);
        *sockFd = -1;
        return -2;
    }

    /* 接続のキュー最大長指定，接続待ち状態へ */
    if(listen(*sockFd, 1) < 0) {
        perror("Error:sock_listen -- listen");
        close(*sockFd);
        *sockFd = -1;
        return -1;
    }

    return 0;
}


/*
 *  sock_accept : 接続待ちしているソケットとの接続が成功したら
 *                リッスン・ソケットを閉じて以降の接続は無視
 */
static int sock_accept(scv_session_t cn, int acceptMax,
                       int acceptNow)
{
    int sod;

    /* 接続されたソケットのファイル・ディスクリプタ取得 */
    if((sod = accept(cn->lstn, NULL, NULL)) < 0) {
        printf("Error:%s -- accept: %s", __func__,
               strerror(__errno()));
        return -1;
    }
    if(acceptMax < ++acceptNow) {
        close(sod);
        return -1;
    }
    close(cn->lstn);
    cn->lstn = -1;

    /*** バッファ・カウンタの初期化 ***/
    cn->sckcnt = cn->sckstp = cn->sckenp = 0;

    return sod;
}
```

リスト4　ネットワーク（LAN）送信処理部

```
/*
 *  buf_to_sock : バッファのデータをソケットへ出力する
 */
static void buf_to_sock(scv_session_t cn)
{
    int res, sdcnt;

    if(cn->ttycnt == 0)
        return;

    sdcnt = cn->ttycnt;
    if((sdcnt + cn->ttystp) > BUFSZ)
        sdcnt = BUFSZ - cn->ttystp;
    if(sdcnt == 0)
        return;
    res = send(cn->conn, &cn->ttybuf[cn->ttystp], sdcnt, 0);
    if(res > 0) {
        cn->ttycnt -= res;
        cn->ttystp = (cn->ttystp + res) % BUFSZ;
    }
}
```

●最初に，データ・バッファや変数の領域を確保し，値を初期化する
●次に，接続待ちのソケットを作成し，使用するデバイス（RS-232-C，LCD）をオープンする

以上で準備が完了し，idleループに入ります．idleループでは以下の処理を行います．

●RFIDリーダ/ライタへRFタグのスキャン・コマンドを発行し，タグの有無と最初（01ブロック）のデータを読み込む
なお，RFIDタグ・スキャン関数を表7に示します．
●ソケットの状態をselect()関数でチェックし，接続要求や，受信データがあれば処理を行う

select()関数は，一度に複数のソケットやファイルの状態を調べることができます．ここでは，接続要求待ちのソケットと，クライアントとの接続が確立したソケットの二つを一度に調べています．

● RFIDリーダ/ライタ制御部の詳細

一番重要なRFIDリーダ/ライタ制御部の説明をします．eBoss-1とRFIDリーダ/ライタの通信シーケンスを図6に示す

リスト5　メイン処理部

```
/************* メイン **************/
int main(int argc, char *argv[])
{
    int             n, res;
    int             tim_f;
    struct timeval  tmv;
    fd_set          fds;
    scv_session_t   scvcn;

    /* 制御端末ハングアップ検出を無視 */
    (void)signal(SIGHUP, SIG_IGN);

    if((scvcn = malloc(sizeof(scv_session_data_t))) == NULL) {
        return EXIT_FAILURE;
    }
    /* シリアル通信バッファ作成 */
    if((scvcn->ttybuf = malloc(BUFSZ)) == NULL) {
        return EXIT_FAILURE;
    }
    /* ソケット通信バッファ作成 */
    if((scvcn->sckbuf = malloc(BUFSZ)) == NULL) {
        return EXIT_FAILURE;
    }

    scvcn->rsfd = scvcn->lstn = scvcn->conn = -1;
    scvcn->bps = DEFAULT_BPS;
    scvcn->csize = DEFAULT_CSIZE;
    scvcn->parity = DEFAULT_PARITY;
    scvcn->stop = DEFAULT_STOP;
    scvcn->flow = DEFAULT_FLOW;
    scvcn->portno = DEFAULT_PORT;
    scvcn->dev = TGT_DEVICE;
    scvcn->ttycnt = scvcn->ttystp = scvcn->ttyenp = 0;
    scvcn->sckcnt = scvcn->sckstp = scvcn->sckenp = 0;


    /* ソケット接続待ちの設定 */
    while((res = sock_listen(&scvcn->lstn, scvcn->portno))
            == -2);
    if(res != 0) {
        return EXIT_FAILURE;
    }

    /* シリアル通信ポートを開く */
    if((scvcn->rsfd = rs_open(scvcn)) < 0) {
        fprintf(stderr, "can't open %s¥n", scvcn->dev);
        return EXIT_FAILURE;
    }

    /* LCDデバイスを開く */
    if((scvcn->lcdfd = lcd_open(scvcn)) < 0) {;
        fprintf(stderr, "can't open '/dev/lcd'¥n");
        return EXIT_FAILURE;
    }

    /* idle Loop */
    for(;;) {
        if(scvcn->rsfd != -1) {
            if(tim_f == 1)
                sleep(1);

            /* RF=ID controll */
            /* Mifare Card read */
            res = mifare_read(scvcn);
            /* card find Sleep */
            if(res == 1) tim_f=1;

            /* icode Card read */
            res = icode_read(0, scvcn);
            /* card find Sleep */
            if(res == 1) tim_f=1;

            /* ISO15693 Card read */
            res = icode_read(1, scvcn);
            /* card find Sleep */
            if(res == 1) tim_f=1;

            if(tim_f == 1){
                tty_to_buf(scvcn);
            }
        }
        /* マスク集合の要素初期化 */
        FD_ZERO(&fds);
        /* select関数で使用する調べたいファイル
           ・ディスクリプタの最大値を初期化 */
        n = 0;

        if(scvcn->lstn != -1) {
            FD_SET(scvcn->lstn, &fds);
            n = scvcn->lstn;
        }
        if(scvcn->conn != -1) {
            /* バッファのデータをソケットへ出力 */
            buf_to_sock(scvcn);
            FD_SET(scvcn->conn, &fds);
            if(n < scvcn->conn)
                n = scvcn->conn;
        }
        if(n == 0)
            continue;  /* ソケットが接続されていないときは
                              以下の処理はしない */

        tmv.tv_sec  = 0;
        /* select関数 のタイムアウトを100msに設定 */
        tmv.tv_usec = 100000;
        /* ソケットが読み書き可能か，*/
        if(select(n + 1, &fds, NULL, NULL, &tmv) <= 0)
            continue;    /* またリッスン・ソケットに接続要求が
                                あるか調べる */

        if((scvcn->lstn != -1)
           && FD_ISSET(scvcn->lstn, &fds)) {   /* 接続受付 */
            scvcn->conn = sock_accept(scvcn, 1, 0);
        }
        /* ソケットがデータを受信したか？ */
        /* ソケットの受信データをバッファへ格納する */
        if((scvcn->conn != -1)
           && FD_ISSET(scvcn->conn, &fds)) {
            sock_to_buf(scvcn);
        }
    }

    return EXIT_SUCCESS;
}
```

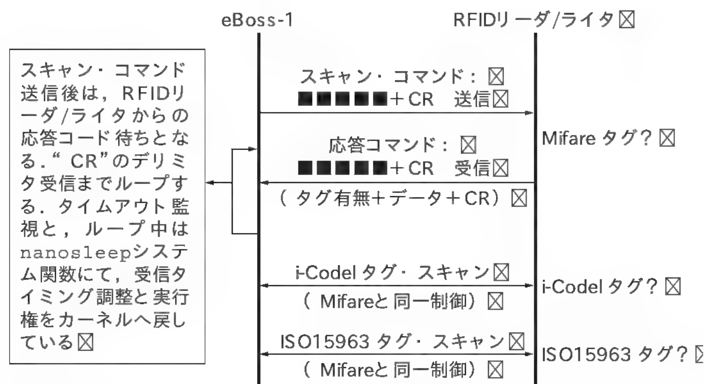


図6 RFIDリーダ/ライタ制御シーケンスの概略

表7 RFタグ・スキャン関数

関数名	機 能
int mifare_read (scv_session_t cn)	Mifareタグ・スキャン関数
int icode_read(int icode, scv_session_t cn)	i-Codel, ISO15693タグ・スキャン関数

ので，あわせてご覧ください．

Mifareタグのスキャン制御ですが，通常は“認証KEY”の登録によりセキュリティ対応が可能です．しかし今回は説明を簡単にするために，認証KEYがないものとして先頭“01ブロック”の読み取り制御を行っています．リスト6がそのプログラムです．処理の概要は，コマンドの送信，応答コードの受信となり応答コードがタグの有無と01ブロックのデータを受信しています．

意外と簡単なプログラムになっていますが，13.56MHzの電

リスト6 RFIDリーダ制御部

```
/* RFID R/W Controll Command & Reply data is NDA.
*'[redacted]' or 'xx'  is secret Command.
*/

/*
 *  mifare_read : Mifare カードの読み取り
 */
static int  mifare_read(scv_session_t cn)
{
    int             res, cnt, pos, cr_f, ret=0;
    char            rs_buf[120], buf[120];
    struct timespec treq, trem;
    int             pre_time;

    /* Mifare read */
    /* RS-232-C 送信バッファにスキャン・コマンド・セット */
    strcpy(rs_buf, "[redacted]¥r");
    rs_buf[xx] = 0x0d;       /* 電文デリミタ  セット */
    res = write(cn->rsfd, rs_buf, xx);     /* 送信  */
    if(res < 0) {
        return -3;
    }

    rs_buf[0] = NULL;
    buf[0] = NULL;
    cnt = 0;
    cr_f = 0;
    pre_time = time(0);    /* 応答待ち 監視時間  set */

    /* RFIDリーダ/ライタ  応答コマンド  受信制御ループ  */
    for(;;){
        /* read Loop */
        res = read(cn->rsfd, rs_buf, xx);
        if(res < 0 ) return -4;
        if(res != 0 || res > 0) {
            /* CR read */
            /* デリミタ受信で完了 */
            if(rs_buf[0] == 0x0d) break;
            if(res == 1) {
                pos=1;
            } else {
                for(pos = 1; pos < res; pos++)
                {
                    if(rs_buf[pos] == 0x0d) {
                        cr_f = 1;
                        break;
                    }
                }
            }
            if(cr_f == 0) {
                strncat(&buf[cnt], rs_buf, res);
                cnt = cnt + res;
                buf[cnt] = NULL;
            } else {
                /* 0x0d */
                strncat(&buf[cnt], rs_buf, pos);
                buf[cnt+pos] = NULL;
                break;
            }
        }
        /* time over ?*/
        /* 応答待ち監視時間  >3 秒間?  */
        if(abs(pre_time - time(0)) > 3)
            return 0;
            /* SLEEP */
            treq.tv_sec  = (time_t)0;  /*1ms スリープ  */
            treq.tv_nsec = 1000000;
            nanosleep(&treq, &trem);
    }
    /* 応答コマンドより RFIDタグの有無を判断し，
       ある時は LCD 表示などをしてソケットでデータ送信 */
    /* RF-ID 有無?  応答コマンド判断*/
    if(strncmp(buf, "[redacted]", xx != 0) {
        /* Mifare LCD 表示  */
        res = write(cn->lcdfd, "Tag = Mifare¥n", 16);
    }
    /* BUZZ on */
    led=7;      /* ブザー，LED  ON  do 制御  */
    write(g_do,&led,1);

    /* Socket buff store Socket data change */
    strcpy(rs_buf, "Mifare   ="); /* LAN Socket 電文作成 */
    strcat(rs_buf, buf);
    buf[0] = NULL;
    strcpy(buf, rs_buf);
    res = strlen(buf);
    buf[res]=0x0d;
    buf[res+1]=0x0a;
    buf[res+2]=NULL;
    res=res+2;

    cn->ttycnt += res;
    strcpy(&cn->ttybuf[cn->ttyenp], buf);
    cn->ttyenp = (cn->ttyenp + res) % BUFSZ;
    ret = 1; /* card read */
  } else {
    ret = 0; /* card read none */
  }
  return ret;
}


/*
 *  ISO15693 : icode-i カードの読み取り
```

波の発信制御，RF タグとの通信などはすべてスキャン・コマンドを受信した RFID リーダ/ライタが制御をしているため，このようになっています．

また，i-CodeI と ISO15963 カードのスキャンは，同一制御でスキャン・コマンドのみ異なるため，同一関数で処理をしています．そして RF タグから読み込んだデータをソケット送信バッファへ格納しています．

6 応用事例

すでに eBoss-1 ボードを実際に搭載し運用をしている例として，トッパン・フォームズ(株)の事例を二つ紹介します．

● 展示会来場者の管理システム

図 7 のように各リーダ/ライタが直接 LAN に接続しているため，リアルタイムで来場者の情報を収集し，1 台のパソコン

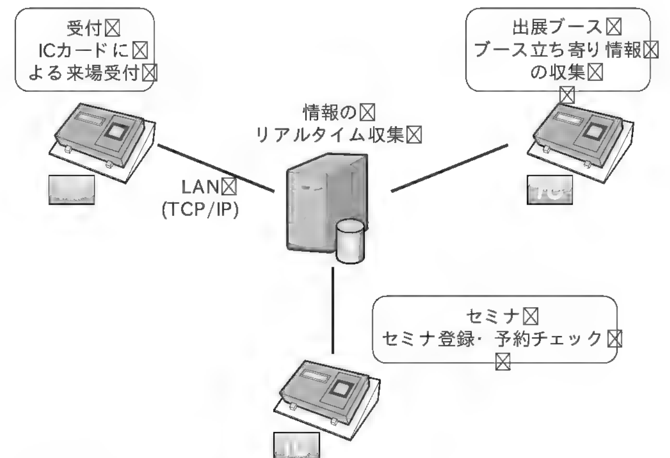


図 7　展示会来場者の管理システムの運用イメージ

リスト 6　RFIDリーダ制御部(つづき)

```
 */
static int icode_read(int icode, scv_session_t cn)
{
    int             res, cnt, pos, cr_f, ret = 0, code;
    char            rs_buf[80], buf[80];
    struct timespec treq, trem;
    int             pre_time;

    /* icode-i card read process */
    code = icode;

    /* icode read */
    if(code == 0){
        /* I-CodeI スキャン・コマンド set */
        strcpy(buf, "[redacted]");
    } else {
        /* ISO15963 スキャン・コマンド  set */
        strcpy(buf, "[redacted]");
    }
    buf[3] = 0x0d;
    /* RFIDタグ・スキャン・コマンド送信  */
    res = write(cn->rsfd, &buf, xx);
    if(res < 0) {
        return -3;
    }

    rs_buf[0] = NULL;  /* 受信バッファ等  初期化  */
    buf[0] = NULL;
    cnt = 0;
    cr_f = 0;
    pre_time = time(0);  /* 応答待ち 監視時間  set */


    /* I-CodeI,ISO15963 RFIDリーダ/ライタ  応答コマンド
       受信制御ループ  */
    for(;;){
        /* read Loop */
        res = read(cn->rsfd, rs_buf, xx);
        if(res < 0) return -4;
        if(res != 0 || res > 0) {
            /* CR read */
            if(rs_buf[0] == 0x0d) break;
            if (res == 1) {
                pos = 1;
            } else {
                for(pos = 1; pos < res; pos++)
                {
                    if(rs_buf[pos] == 0x0d) {
                        cr_f = 1;
                        break;
                    }
                }
            }
            if(cr_f == 0) {
                strncat(&buf[cnt], rs_buf, res);
                cnt = cnt + res;
                buf[cnt] = NULL;
            } else {
                /* 0x0d */
                strncat(&buf[cnt], rs_buf, pos);
                buf[cnt+pos] = NULL;
                break;
            }
        }
        /* time over ?*/
        if(abs(pre_time - time(0)) > 3)
            return 0;
         /* SLEEP */
         treq.tv_sec = (time_t)0;
         treq.tv_nsec = 1000000;
         nanosleep(&treq, &trem);
    }
    /* 応答コマンド解析  */
    if(strncmp(buf, "[redacted]", xx) != 0
        && strncmp(buf, "[redacted]", xx) != 0) {
        if(code ==0) {
            /* LCD表示  */
            res=write(cn->lcdfd, "Tag =iCode I¥n", xx);
            strcpy(rs_buf, "iCode I  =");
        } else {
            res=write(cn->lcdfd, "Tag=ISO15693¥n", xx);
            strcpy(rs_buf, "ISO15693 =");
        }
        /* BUZZ on */
        led=7;       /* ブザー, LED  ON  do制御  */
        write(g_do,&led,1);

        /* Socket buff store Socket data change */
        strcat(rs_buf, buf);  /* LAN Socket電文作成 */
        buf[0] = NULL;
        strcpy(buf, rs_buf);
        res = strlen(buf);
        buf[res] = 0x0d;
        buf[res+1] = 0x0a;
        buf[res+2] = NULL;
        res = res + 2;
        cn->ttycnt += res;
        strcpy(&cn->ttybuf[cn->ttyenp], buf);
        cn->ttyenp = (cn->ttyenp + res) % BUFSZ;
        ret = 1; /* card read */
    } else {
        ret = 0; /* card read none */
    }
    return ret;
}
```

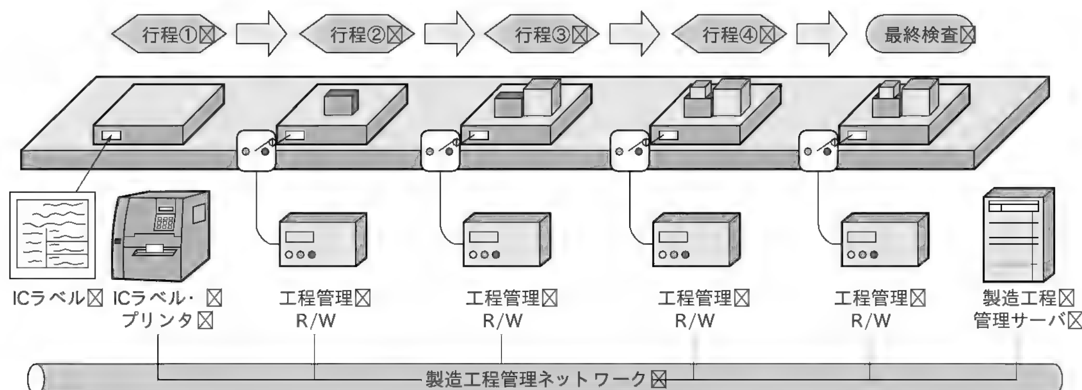


図8
製造工程管理システムの運用イメージ

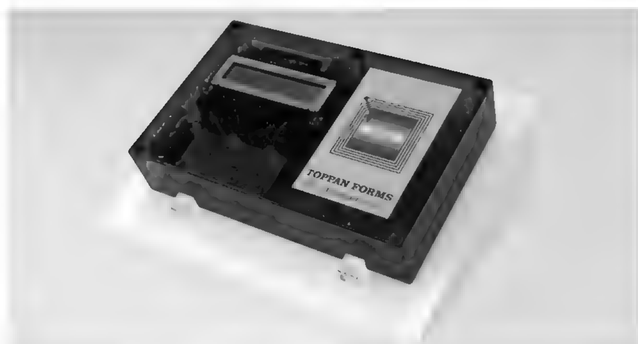


写真4　展示会来場者の管理システム用のRFIDリーダ/ライタ〔トッパン・フォームズ(株)提供〕

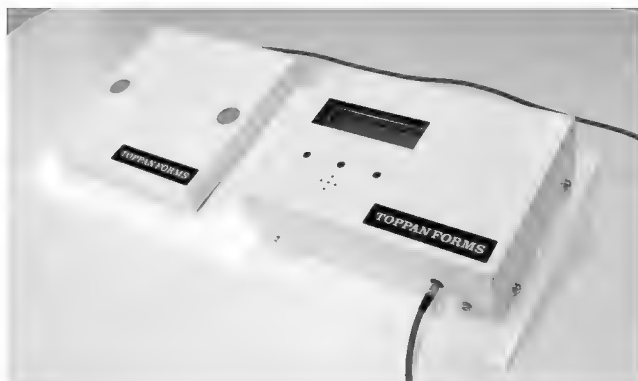


写真6　製造工程管理システムのRFIDリーダ/ライタ〔トッパン・フォームズ(株)提供〕

での情報の一元管理が可能です(写真4，写真5)．

おもな機能としては，タグの読み取り，ソケット通信(LAN)，LCDへの状態表示，タグ読み取り時にLEDやブザーを点灯する，などがあります．

●製造工程管理システム

図8のように各工程でRFタグの情報を読み取り，ネットワークを介してサーバへ情報を送信します(写真6)．そのため，各工程では正しい手順を経ているか否か，各工程内の生産量をリアルタイムに把握することができます．また同時に，ログ情報をCFカードに保存しています．

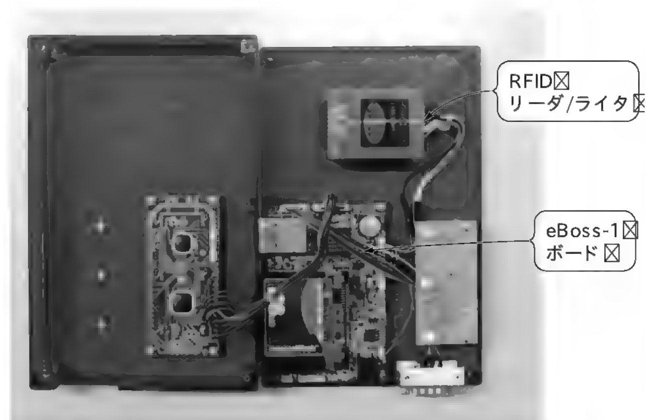


写真5　RFIDリーダ/ライタの内部

おわりに

eBoss-1については，今回の記事を多くの方に実体験できるように，本ボードの低価格品(LAN 1チャネル，RS-232-C 1チャネル，DI/O各4点)を用意し，開発リソースをセットにしたものを現在，準備しています(詳細は，Webページにて公開予定．http://www.cht.co.jp/)．

今回は，RFIDリーダ/ライタへの利用例ですが，eBoss-1はユビキタス社会における各種センサ，監視機器，ネットワーク通信機器として幅広く利用できるよう，展開していく予定です．

短期的にはMMUレスのCPU市場への展開を狙っており，16ビット/32ビット MMUレスのCPUへ順次対応します(SH-2：2004年11月リリース予定)．また，中長期的には今後，加速度的に普及が始まるシステムLSI(SoC，SIP，ASIC)への搭載のための準備をすすめています．

まつなが・たかふみ
コンピュータ・ハイテック(株)エンベデッドシステム事業部
E-mail：eboss@cht.co.jp

電波を発するタグによる
病院・介護ホーム支援システムの実例

6 アクティブRFIDによる所在管理システムの実際

馬場 功



ここまでの章ではRFIDリーダ/ライタが電波を発し，それに対して(バッテリをもたない)RFIDが返答するという「パッシブRFタグ」について解説してきた．本章ではバッテリを搭載したRFID側が電波を発する「アクティブRFタグ」について解説する．アクティブRFタグはバッテリを搭載していることから，バッテリ切れの対応策なども取られている．アクティブRFタグを実際に使うにあたって必要となるこれらの知識についても紹介する．

また，すでにアクティブRFタグはさまざまな分野において実用化されている．今回は電波を発するタグによる病院・介護ホーム支援システムの実例を紹介し，アクティブRFタグの用途を模索する．

(編集部)

1 アクティブRFIDシステム

● アクティブRFIDとはなにか

アクティブRFIDとは，電池を搭載して自律発信を行い，IDを読み取る際にゲート型の読み取り装置を設置する必要がない，送信機(タグ)から受信機へ一方通行の通信しか行わないRFIDです．アクティブRFIDシステムの例として，Local Area Search〔(株)キュービックアイディ〕を取り上げ，解説します．Local Area Searchは，300MHz微弱電波帯を利用し，通信距離が半径約7mのローカル・エリアにある電池を搭載した送信機(タグ)のIDを検出し，特定できます．IDを間欠自動発信する送信機(タグ：LAS300T)と，送信機(タグ)からのIDを受信する受信機(LAS300R：写真1はLAS300R-DVダイバシティ・タイプ)とから構成されています．

基本の動作としては，各受信機が検出エリアを持ち，みずからIDを送信するタグが，受信機の検出エリアの中に入ると，そのIDを受信して，受信した情報を，ネットワークを通して，上位へ送るしくみになっています．

● Local Area Searchの特徴

▶ タグ個別に受信信号強度の取得が可能

通常，アクティブRFIDは，エリア内でのタグの有無の検出を基本としていますが，Local Area Searchは，受信信号強度も同時に検出するという特徴をもっています．

受信信号強度については，タグごとに検出することができ，タグが受信機に近づくと受信信号の強度が上がり，遠ざかると受信信号の強度が下がります．

アクティブRFIDのトライアル・キットに入っている評価ソフトの画面(図1)は，グラフの縦軸が受信信号の強度，横軸が時間となっています．これが赤い線の場合は，タグが受信機にだんだん近づいてきていることを表しており，それとは逆に，黄色い線は，タグが受信機から遠ざかって行っていることを示しています．これによって，タグがおおまかに，受信機に対して遠くにあるのか近くにあるのかがわかり，タグと受信機の相対的な位置関係を想定することが可能になります．

▶ 受信機は，システム拡張性が高い設計になっている

受信機は，10Base-Tのコネクタがついており，LAN接続が可能になっています．このため，LANが走っている環境におい



写真1 LAS300R-DVダイバシティ・タイプ

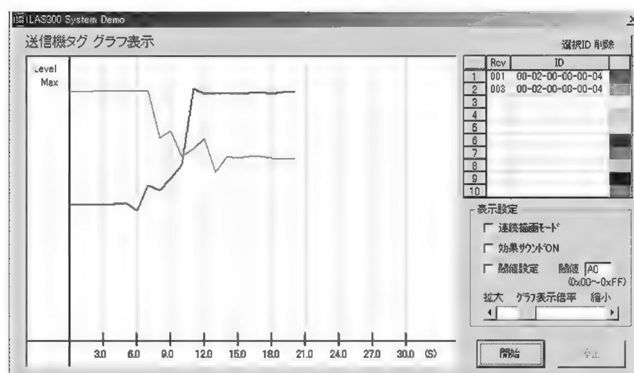


図1 トライアル・キットに入っている評価ソフトの画面

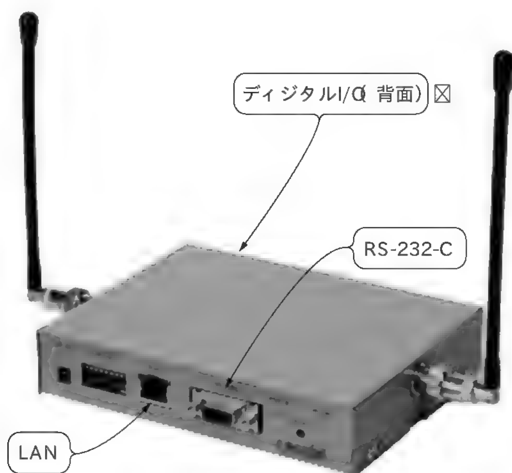


写真2　受信機の端子

ては，受信機の設置に大規模な工事が必要なく，簡単に設置することができます．

また，シリアル通信用に，RS-232-Cのコネクタが付いており，PCにつなげることでスタンドアロンでの利用も可能になっています．

背面には入出力4ビットずつのディジタルI/Oポートが用意されており，入力側には自動ドア・センサ，出力側には警告灯といったような外部機器と連動したシステムを構築することが可能です(写真2)．

▶ タグは，IDの送信間隔の変更が可能

一方通行の発信しか行わないアクティブRFIDは，つねに電波を使ってIDを発信し続けなければなりませんが，それに必要な送信間隔はアプリケーションによって大きく異なります．

ここで使うタグの送信間隔は，標準で0.5秒です．それ以外に，0.2秒から15秒までトータル8通りの送信間隔がプリセットされており，ジャンパ線をカットすることで，ユーザが簡単に送信間隔を変更することが可能になっています．したがって，ユーザーが使用するアプリケーションにおいて，どれくらいの送信間隔がもっとも適しているのかを検討することが容易にできるようになっています．

▶ 電池残量警告が取得できる

アクティブRFIDの最大の欠点は，電池寿命があるため，電池交換を行わなければならないということです．しかも，電圧が下がることは，タグの誤動作の原因にもなりかねないため，電池残量を何らかの形で認知することと，電圧低下にともなう誤動作対策が必要となります．

そこで，ここで使うタグは，一定のレベルに電圧が低下すると，IDに加えて電圧の低下を警告する警告情報も送信し始めます．これによって，タグごとに電池交換の時期を知ることができます．また，電圧低下に伴う誤動作防止のため，電圧低下の警告情報が出てから一定期間経つと強制的に送信を止める機能も付いています．

▶ 受信エリア内の複数のタグの読み分けが可能

複数のタグが，受信機の検出エリア内に存在する場合，一つの受信機でどれだけのタグが検出できるかによって，アプリケーションの負荷が変わってきます．この場合，送信間隔によって，読み分け数は異なりますが，最大500個のタグの読み分けが可能になっています．

● ネットワーク接続を利用した応用例

このLocal Area Searchを使って，ネットワーク接続を利用した応用例を紹介します．

たとえば，図2に示すように，タグを持った人が受信機Aのエリアに入ると，受信機AがタグのIDを受信し，そのID情報をシステムの上位に送ります．このことによって，そのIDを持った人が，受信機Aの検出エリアの中に入ったことが認知できます．

次に図3のように，そのタグを持った人が，受信機Aのエリアから受信機Bのエリアに移ったとします．この場合，タグを持った人のタグのIDは受信機Aでは検出できなくなり，受信機Bから検出されるようになります．このことによって，そのIDを持った人が，受信機Aの検出エリアから，受信機Bの検出エリアに移動したことが認識され，このタグを持った人の導線検出を行うことができます．

それでは，図4のようにタグを持った人のIDが，受信機Bと受信機Cから受信されるような場合，どのようになるかを考えてみます．タグのIDは受信機Bと受信機Cから受信されま

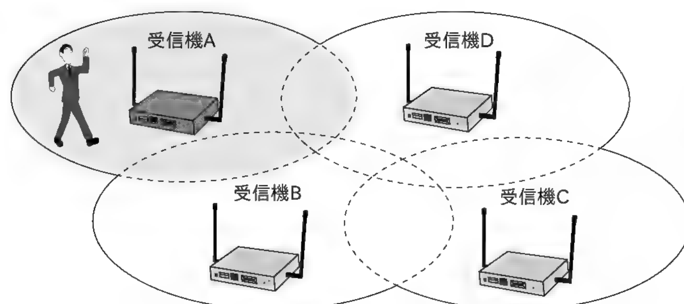


図2　タグを持った人が受信機Aのエリアに入る

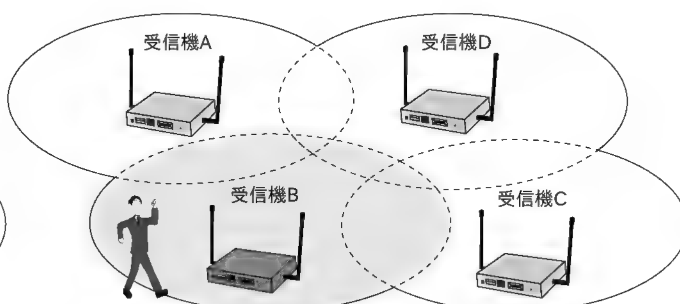


図3　受信機Aのエリアから受信機Bのエリアに移る

すが，それとともに，それぞれ受信信号強度も検出しています．それによって，それぞれの受信信号強度の強さから，相対的に受信機Bに近いのか，受信機Cに近いのかが判別でき，おおよその距離を推定することが可能になります．

これが，図5のように，受信機Dも含めた，3台の受信機がIDを受信できるような場合はどのようになるかを考えると，今度は受信される受信機の数が3台になるため，3点からの相対的な受信強度を測定することによって，3点測量から，タグを持った人のおおよその所在地を推定することが可能になります．このことにより，導線検出も，より細かい範囲で検出することができます．

2 アクティブRFIDを使ったシステムの事例

● 病院・介護ホーム支援システムの実例

このネットワークを使って人の所在管理を実際に導入しようとしている実例が，(株)コム・マックス(愛知県名古屋市守山区)が開発し，医療法人フジタ(愛知県名古屋市東区)が導入する，老人介護施設においての入所者の位置検出，安全管理を行う，病院・介護ホーム支援システムです．医療法人フジタは，愛知県内で病院並びに介護老人保健施設を運営している医療法人で，現在と愛知県内の介護老人保健施設フジオカ(愛知県西加茂郡藤岡町)を運営しています．

今回導入される病院・介護ホーム支援システムは，2005年春に愛知県名古屋市緑区にオープンする予定の介護老人保健施設への導入が決まっており，現在介護老人保健施設フジオカで導入のための実証実験を行っています．

病院・介護老人ホーム支援システムの内容は，次のようになっています．

▶ システムの目的

システムのおもな目的は，以下のようなものです．

1) 入所者・スタッフの居場所を検知する
2) IDごとにレベル指定を行うことにより，入所者・スタッフの立ち入り制限を行う
3) IDから，入所者・スタッフの居場所を検索する
4) 危険エリアや死角に入所者が入るとナース・センタに危険信号を通知し，危険を未然に防ぐ
5) 将来的には，PDAとの連動により，入所者データの連携を図る

システムの概要は，エリア検出システムとPDAカルテ参照システムからなっており，それぞれ次のような内容になっています．

● エリア検出システム

1) 動作概要

エリア検出システムの動作概要は，入所者・スタッフ外来者にタグを持たせ，各人の所在，部屋の出入り，安否の確認などを行うもので，具体的には以下のとおりです．

- タグごとにIDと個人情報(名前など)を結びつける
- 最後に会った人(人名・時間)を特定する
- 入所者，職員，外来者(出入り業者，面会者など)の区別を行う
- 色(赤，黄，青など)により危険度を認識する

2) 基本動作

基本動作としてはアクティブRFIDシステムを使って，次のことを行っています．

- 各受信機(LAS300R)が，どのIDを受信し，どのIDを受信していないかを確認する
- 受信機が，タグ(LAS300T)のIDを検知したときに，その受信機のエリアにそのタグを持った人が入ったと判定する
- 複数の受信機が一つのIDを検知したときは，受信信号の強度が強い受信機のエリアにそのタグを持った人がいるものとする

この基本動作を確保するため，各受信機はソフトウェアで感度を調節し，検知エリアの調整を行います．

● システムの基本機能

それでは，この動作概要と基本動作をもとに，システムとしてどのような基本機能をもっているかを説明します．

1) タグ所持者の位置情報を平面図上に表示(図6)

タグごとに，平面図上のどの位置(どのエリア)にタグを持った人がいるかを認知し，マークおよび名前を表示する．

図6は，色ごとに，入所者，スタッフ，外来者を区別し，それぞれどの部屋に現在いるかを表示しています．

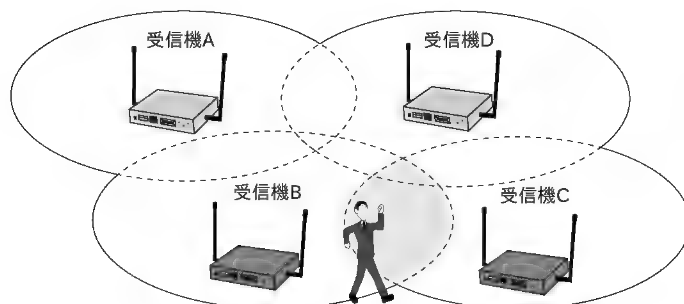


図4　受信機Bと受信機Cから受信されるような場合

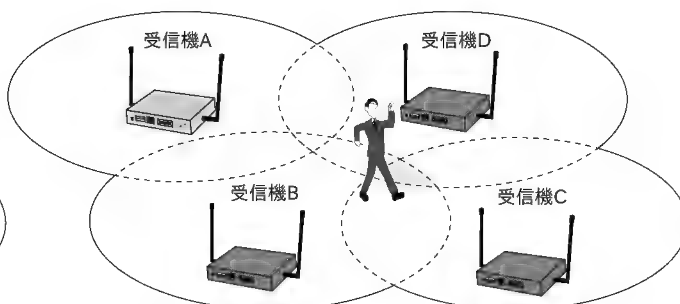


図5　3台の受信機がIDを受信できる場合

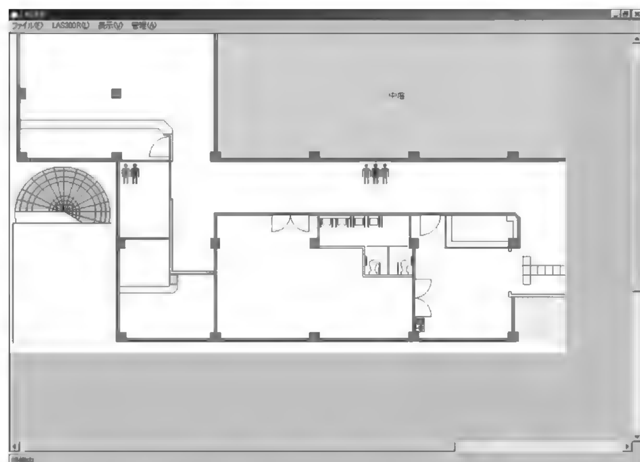


図6　タグ所持者の位置情報の平面図上表示

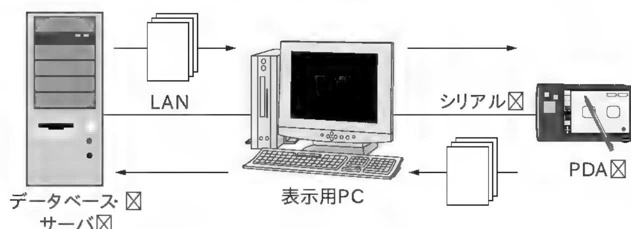


図7　PDAカルテ参照システムの動作概要

2) 指定レベルのタグ所持者の位置情報の平面図上表示

タグにつけられたIDごとにレベル設定を行い，ある指定レベルのIDがつけられたタグを持っている人が現在どこにいるかを表示します．

3) タグ所持者の検索，平面図表示

名前を指定することにより，タグのIDと個人情報の関連付けから，その人が現在いる場所を特定し，平面図上でどのエリアにいるかを表示します．

4) 警報時アラート表示

危険エリアに権限のない人が入り込んだのを検知した場合は，関連平面図をトップに表示し，アラート表示をします．

5) 警報履歴検索表示

そのときまでに発報された警報の履歴を絞り込み検索して，警戒エリアの入室履歴を見ることができます．

6) エリア内タグ所持者の名前表示

指定エリアにタグを持った人が複数いた場合，平面図表示上では名前が判別できなくなる場合があるため，別の表示枠に，指定されたエリアに現在，だれがいるのか，名前を列挙します．

● PDAカルテ参照システム

PDAカルテ参照システムは，エリア検出システムと共通のデータベース・サーバから，PDAに入所者のカルテをダウンロードし，入所者の容態を入力し，データベース化して入所者の健康管理に使用するシステムです．

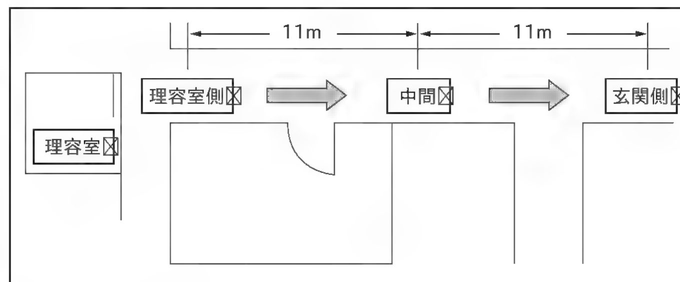


図8　テスト1の条件

具体的な動作概要は図7のようになっています．

1) データベース・サーバから入所者ごとのカルテをPDAにダウンロードする．ダウンロードするときは，表示用PCで操作する
2) 健康チェック・データをPDAに入力する
3) 健康チェック・データをPDAからデータベース・サーバにアップロードすることにより，健康チェック・データをデータベース化する

● 将来的なシステム統合

将来的には，このエリア検出システムとPDAカルテ参照システムを統合し，PDA用の受信機(スキャナ)で入所者が所持しているタグからIDを読み取り，PDAにダウンロードしたカルテを表示させ，入所者の誤認をなくしたり，健康チェックができなかった入所者や，処置が必要な入所者の位置情報を平面図上に表示するようなシステムの拡張を考えています．

● システム機材

このシステムで使用される機材は，1フロアあたり，下記のようになっています．

- データベース・サーバ：1台
- 表示用PC：1台
- Local Area Search受信機(LAS300R)：44台
- Local Area Searchタグ(LAS300T)：100台
- PDA：10台

今回の病院・介護ホーム支援システムで，もっとも重要なことは，Local Area Searchの受信信号強度データが，位置検出をするにあたり，どの程度使用できるかを確認することにあります．

そこで位置検出システムの導入にあたり，次のような4種類のテストを行い，数多くの受信信号強度データを取得しました．

● テスト1の実施と結果

はじめに，静的精度を確認するため，ダイバーシティ・タイプ(受信アンテナ2本)の受信機(LAS300R-DV)を実際に図8の場所に設置し，タグの受信信号強度(電界強度)を測定しました．

▶ テスト条件

- 廊下は中心を歩く
- 各受信機が受信した受信信号強度(電界強度)を測定する

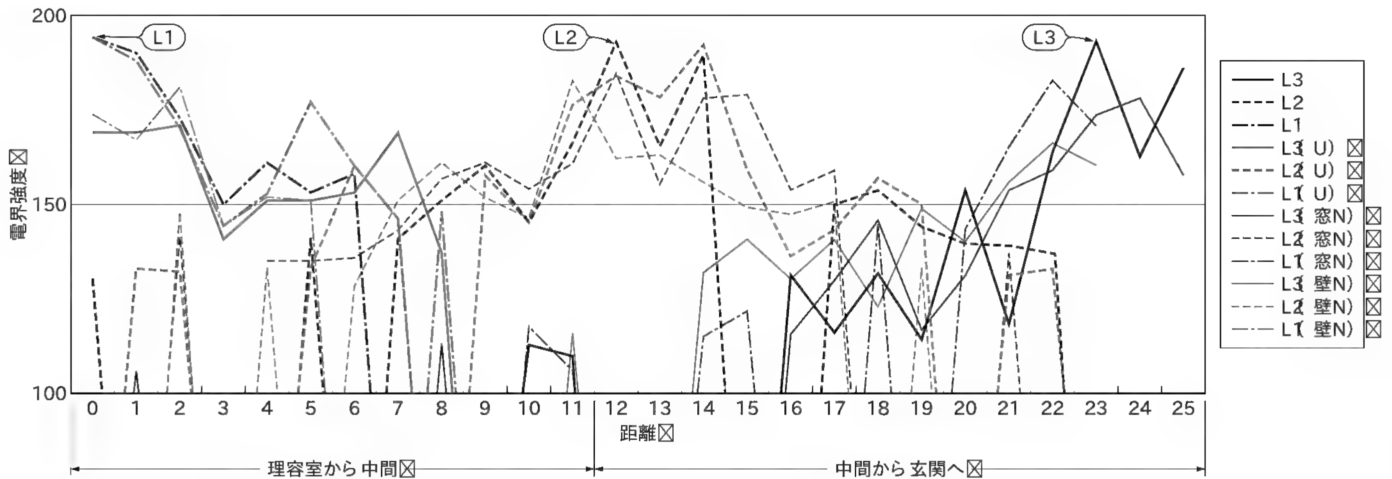


図9　テスト 1の結果

▶ テスト方法

- 理容室側(L1)から中間(L2)，玄関側(L3)へと歩き，1m 間隔で受信信号強度(電界強度)を測定する
- タグの電池がついているほうを正面として，タグの方向による受信信号強度(電界強度)の測定を行う
- タグの方向は，正面→裏面(U)→左向き(窓 N)右向き(壁 N)の順にテストを行う．

▶ テスト結果

この実績を取ったデータが，図9です．

図9のデータを見ると，受信機とタグの距離が長くなれば，受信が不安定になる傾向は見られるものの，半径 7m の範囲においては，タグの方向に大きく依存することなく，おおむね，距離に応じた，受信信号強度データが取得できることがわかります．

● テスト 2 の実施と結果

部屋と廊下において，受信機(LA S300R)を 3 台配置し，部屋に入ったこと，室内から出たことを受信信号強度(電界強度)で判断することができるかどうか，図10の場所において，次のような条件，方法でテストを行いました．

▶ テスト条件

- 廊下は中心を歩く
- 受信機は理容室側，玄関側の間隔は 11m とする

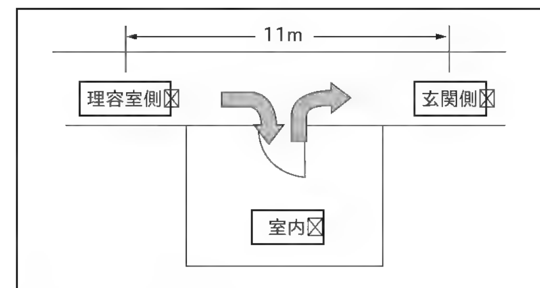


図10　テスト 2の条件

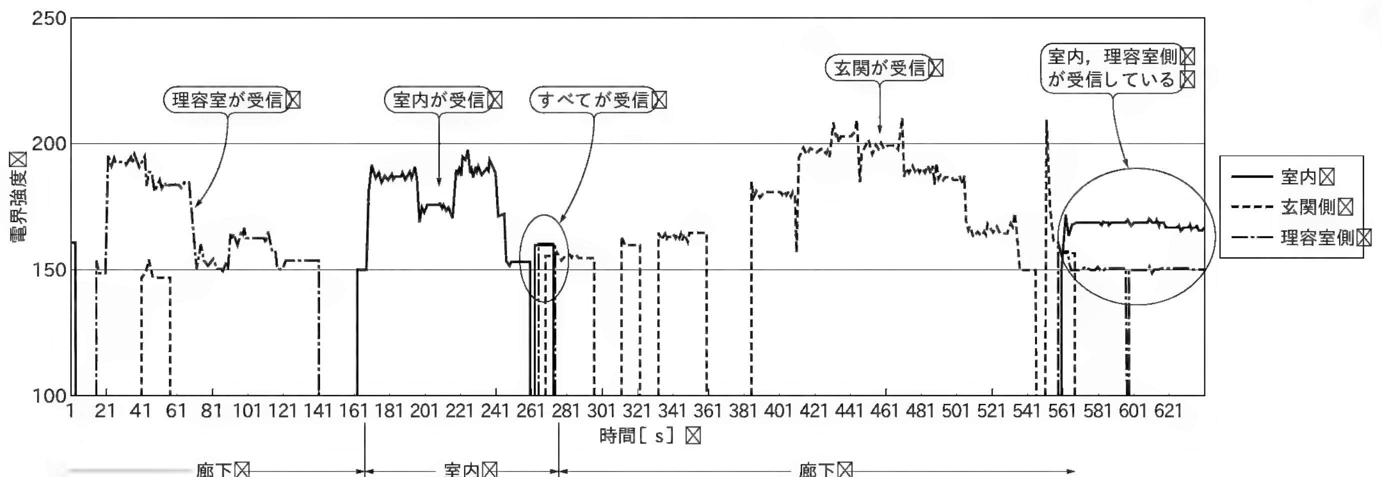


図11　テスト 2の結果

●部屋の扉は開放とする
●室内の受信機の位置は，扉の正面で距離は3mとする
▶テスト方法
●理容室から室内へ歩く
●室内から玄関へ歩く

このときの受信信号の強度(電界強度)を，時間単位(1秒)で取り込む.

▶テスト結果

この実績を取ったデータが，図11です.

図11のデータを見ると，入室，退出の際に，一部にどの受信機も感知しない場合，およびすべての受信機が同じくらいの受信信号の強度でIDを受信する場合が存在したものの，入室，退室のだいたいの傾向は読み取れることがわかりました.

● テスト3の実施と結果

テスト2と同様に，部屋と廊下において，受信機(LAS300R)4台配置し，室内から出た後，長い廊下をまっすぐに歩いて行った場合，部屋から出たことと，廊下を歩いていったときの位置情報が取れるかどうかを，図12の場所において，次のような条件，方法でテストを行いました.

▶テスト条件
●廊下は中心を歩く

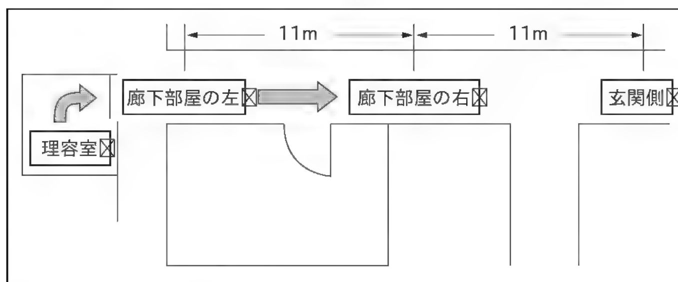


図12　テスト3の条件

●廊下部屋の左と廊下部屋の右の受信機の間隔は11mとする
▶テスト方法
●理容室から廊下へ出る
●廊下を玄関に向けて歩く
●このときの受信信号の強度(電界強度)を時間単位(1秒)に取り込む
▶テスト結果

この実績を取ったデータが，図13です.

図13のデータからは，理容室内において，廊下部屋の左の受信機もIDを受信するものの，理容室の外に出ることによって，理容室内の受信が終わり，玄関側に向けて，徐々に廊下部屋の左側の受信信号強度が下がり，廊下部屋の右側の受信信号強度が強くなっていくのがわかります.

● テスト4の実施と結果

長い廊下に受信機(LAS300R)を3台配置し，長い廊下を歩いていったときに，人が移動しているということを，設置した受信機で捕らえることができるかどうか，図14の場所において，次のような条件，方法でテストを行いました.

▶テスト条件
●廊下は中心を歩く
●廊下下手の左と廊下部屋の右，廊下部屋の右と玄関側の受信機の間隔はそれぞれ11mとする
▶テスト方法
●廊下部屋の右から玄関側に向って歩く
●玄関側を通り越し，先へ歩く(Uターン)
●このときの受信信号の強度(電界強度)を単位時間(1秒)に取り込む

—テスト結果

この実績を取ったデータが図15です.

図15のデータからは，廊下に設置された3台の受信機で，移動している人の持っているタグの受信信号の強度が順次移って

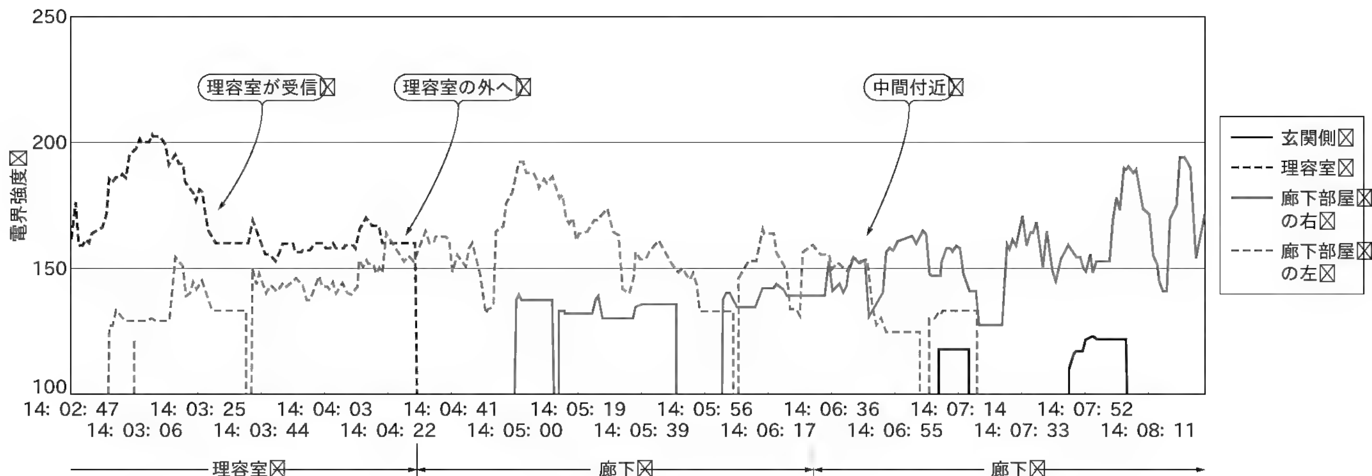


図13　テスト3の結果

いくようすがきれいにわかります．玄関側の先まで行った場合は，最終的には取れなくなることもこの図からわかります．

以上のような，実際の設置場所による各受信機（LAS300R）の受信信号強度データの蓄積により，全体的な受信機の設置場所の検討に入っていきます．図16の受信機の配置図は，これらの受信信号強度データに基づいて，死角をできるだけなくすように配置したものです．

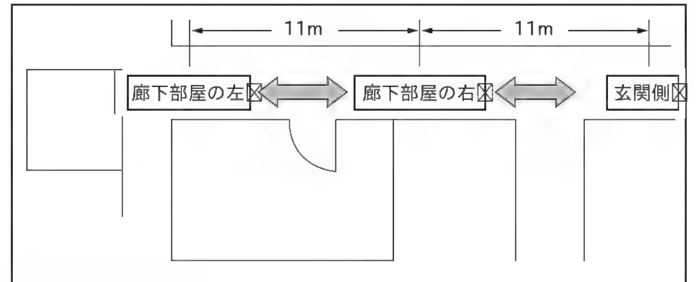


図14　テスト 4の条件

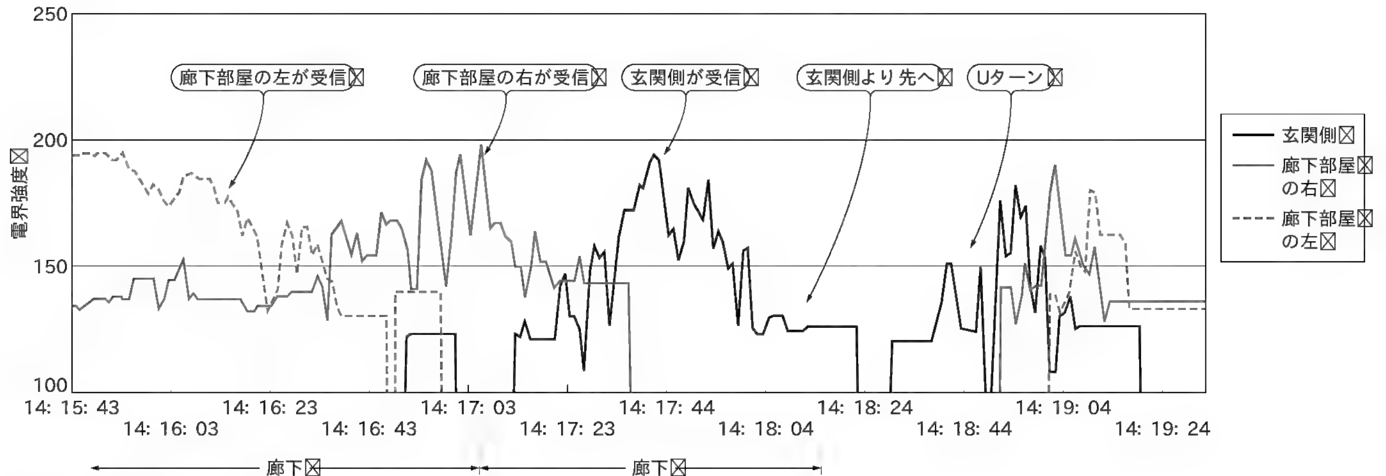


図15　テスト 4の結果

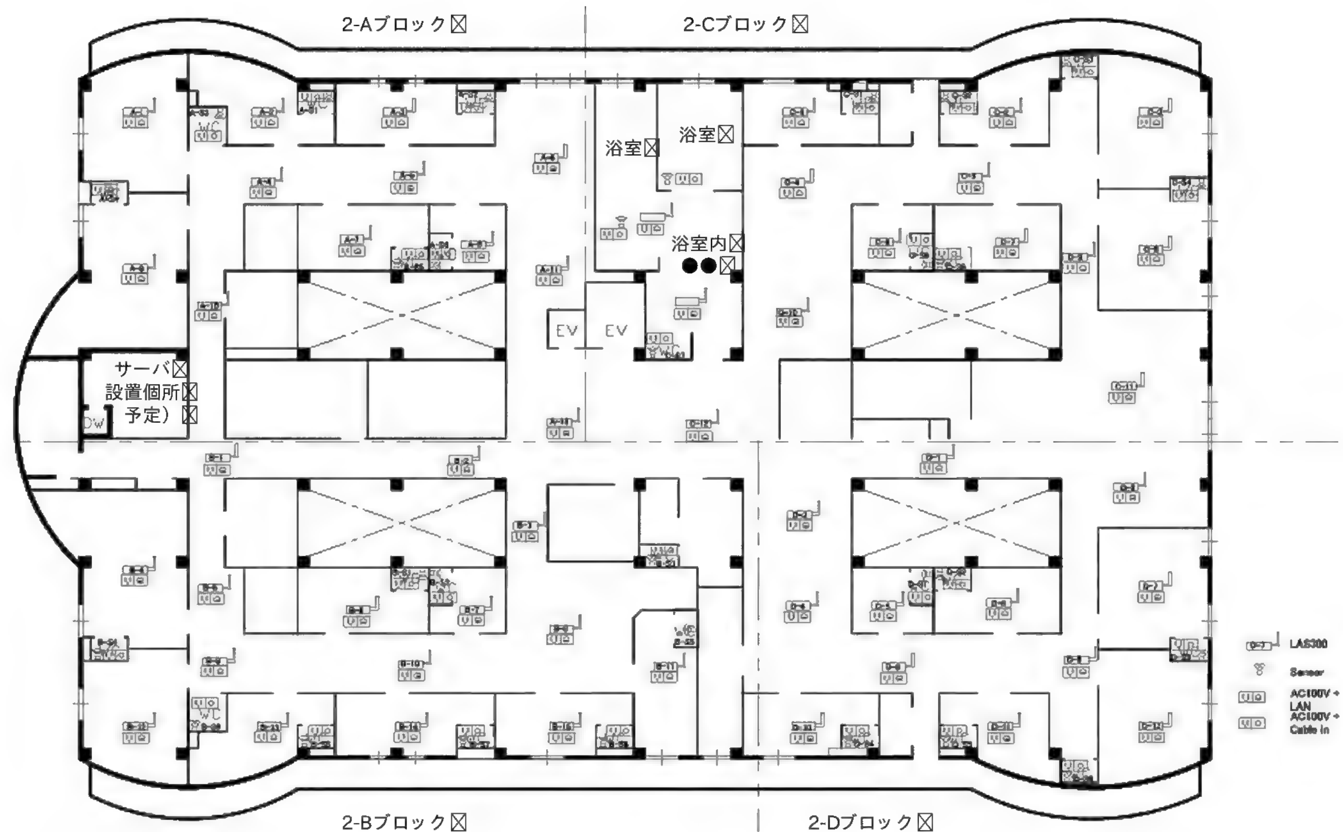


図16　実際の受信機の配置図

3 その他のシステム事例

今回紹介した事例は，人がタグを持つ事例ですが，アクティブRFIDシステムの実際の利用シーンでは，タグを物につけることが圧倒的に多いのが実情です．その例をまとめたのが図17です．これらの例は，すべて実際に使われている，または，導入に向けての実証実験を行っている案件です．

● フォークリフトの入退出管理

倉庫におけるフォークリフトの入退出の管理に使われています．フォークリフトにタグをつけるのも，人がタグを持つのも基本的に同じですが，ディジタルI/Oを利用して，シャッタとの連動を図っています．フォークリフトだけでなく，人もタグを持っているので，システム上で，人とフォークリフトの違いを認識し，タグごとに，シャッタの開閉が必要かどうかの判断を行っています．

● 在庫管理・商品管理

アクティブRFIDシステムを使っての在庫管理・商品管理は，数量管理というよりは，在庫品・商品の所在管理という意味合いでの使い方が主流です．自動倉庫など，大規模な自動倉庫システムを運用している場合は必要ありませんが，在庫品を棚に置いて管理しているような倉庫では，自動倉庫と違い，いくらコンピュータを使っているといっても，必ず人手を介するため，置き場所が違ったり，誤入力などのまちがいを経験している方も多いと思います．

このような場合への対応として，アクティブRFIDシステムの利用によって，タグをつけた商品が，どの受信機の近くにあるかを認識し，商品を探すエリアを限定することによって，商品を探す時間の短縮を図っている会社があります．

● 製造部品・工具などの管理

この場合の製造部品は，金型などの製造設備に付随して，ユーザの注文ごとに製造設備のパーツを取り替えなければならないような場合の，製造パーツのことです．このような製造部品は，必ず何かの部品が機械に使われているわけなので，すべての部品を保管する場所を確保しているわけではありません．

しかし，製造製品の切り替えにともない製造パーツの変更を行うときは，その製造パーツの保管場所がすぐわかり，変更時間のタイムロスをなくすことが，機械の稼働率をアップさせるうえで重要になってきます．規模が小さなうちは，機械の近くに保管棚などを設けて人的管理で十分に対応できますが，規模の拡大とともに，人的管理の限界が出てきます．加えて，設備を共通化して，複数台の併用となるとますます管理が難しくなり，稼働率のダウンという問題が出てきます．

その解決策として，保管場所ごとに受信機を設置し，製造部品にタグをつけることによって，製造部品の所在管理と，製造切り替えの時間短縮を図っている会社があります．

● 毒劇物管理

近年，報道でも取り上げられていますが，管理者が知らない間に毒劇物がなくなり，管理責任を問われているケースが見られます．

ある企業では，毒劇物の保管容器にタグ（LAS300T）を付けて毒劇物の管理を行っています．

毒劇物の管理は，当然のことながら，専用保管庫を持ち，保管庫への入場者制限を行っていますが，持ち出す毒劇物の管理が人的に行われている場合は，帳簿上で持ち出したものと，実際に持ち出したものの照合が必ずしも合っているとは限りません．そこで，アクティブRFIDシステムの受信信号強度の変化情報をもとに，入場した人がどの容器を触ったかという情報を把握しています．通常，このような保管場所は，頻繁に人の出入りがあるわけではないので，容器に付けられたタグから発信される電波の受信信号の強度は安定したレベルを保ちます．しかしながら，これを少し動かしただけでも，受信信号の強度のレベルが変動するため，受信信号の強度のレベル変動を見ることによって，今何が入っている容器をだれが触ったかを，入場者情報と連動させることが可能になっています．

● 固定資産管理

ここでいう固定資産とは，土地，建屋，生産設備といったような大型固定資産ではなく，持ち運びが可能であるが，金額的には固定資産登録し，いろいろな人が使いまわすような，技術における測定器のような設備のことです．

このような設備は，生産に使う設備と違って，毎日稼動しているわけではなく，また，特定の人が専用に使っているわけでもありません．保管部門，保管責任者はいても，管理が十分に行われていなければ，棚卸しになって，現品がどこに行ったかわからなくなっているという経験のある方も多いと思います．そこに，アクティブRFIDシステムを使って，そのような設備が今どのエリアにあるかを監視し，不明資産の防止を行っている会社もあります．

以上，実際の使用例を列挙しましたが，これらのアプリケー

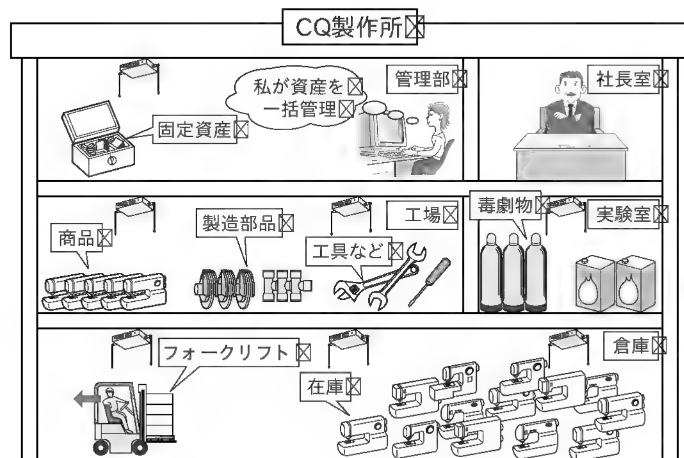


図17　アクティブRFIDシステムの実際の利用

ションに共通していえることは，すべてタグの利用について，再利用可能なアプリケーションであり，個別にクローズしたシステムとして稼動している内容だということです．

4 アクティブRFIDの利用にあたって

最後に300MHz微弱電波帯のアクティブRFIDを利用したシステム構築に当っての留意点を述べておきたいと思います．

● 微弱電波帯の特性を知ってシステム構築をする

微弱電波は，システム事例の中で受信信号の強度のデータを示したように，必ずしも一定のレベルを保って受信できるものではありません．受信機の設置場所，周りの環境の変化などで電波環境は大きく変わります．そこで場所ごとの環境を見極めたうえで，設置場所，対応策を考えていく必要があります．そのためには，実際のデータに基づいた検証が必要です．

● 機器の特性を十分理解したうえでシステムを構築する

Local Area Searchには，先に述べたように数多くの特徴をもっています．今回紹介したシステムも，受信信号の強度の取得機能やデジタルI/Oポートの利用など，製品特性をうまく利用したシステムが数多く含まれています．これらの機能をいかにうまく使ってシステムを構築するかがアプリケーションの良し悪しを決めます．

これらの特性を十分に理解しないままシステムを構築すると，実際の使用局面になって，誤動作などのため，実用的でないシステムになっていることがあります．

おわりに

アクティブRFIDは，一般的なパッシブRFIDと異なり，IDを無意識に認知する，あるいはIDを読み取るのに何らかの行為または，構造物を必要としないということをベースとした市場ニーズに基づき，パッシブRFIDとは異なった市場を形成していくものと考えます．

そのためには，今後より一層，微弱電波帯とアクティブRFIDシステムの特性を熟知したアプリケーションの開発に向けて，各方面からの協力を得られたらと思っております．

今回の原稿執筆にあたりご協力いただいた，(株)コム・マックス，医療法人フジタ，介護老人保健施設フジオカ，(株)エヌアイディに感謝いたします．

ばば・いさお　(株)キュービックアイディ 取締役RFID事業部長

1 2 App 3 4 5 6



バーコードよりも大容量，高密度で誤り訂正機能を持つ

二次元コード

QRコードの概要

末武 陽一

1 QRコードの成り立ち



物流などの分野で商品の識別を行うためにバーコードが普及しているが，近年，このバーコードよりも大容量のデータを高密度で表現する「二次元コード」が使用されるようになってきている．

一次元コードであるバーコードは水平方向だけに情報を持つが，二次元コードは水平方向と垂直方向の二方向に情報を持つので，バーコードよりも飛躍的に大きな容量のデータを扱える．また，バーコードに比べて同じデータを扱う場合に印字する面積が小さい(高密度である)，360°どの方向からでも読み取れる，誤り訂正機能をもっているので少々の汚れや欠けがあっても読み取ることができる，などの利点がある．

代表的な二次元コードとしては，「QRコード」や「PDF417」，「DataMatrix」，「MaxiCode」などがあり，そのほかにもさまざまな規格がある．本稿では，この二次元コードのうち，規格が公開されており，最近ではコンシューマの分野からも注目されつつある「QRコード」について取り上げる．

● 二次元コード「QRコード」

QRコードは，1994年に日本電装株式会社〔現在の(株)デンソー〕が開発したマトリックス型の二次元コードである(図1)．JIS X0510(1999)，ISO/IEC18004(2000)として規格が公開されており，自由に使用することができる[注1]．

QRコードの「QR」とは「Quick Response」の略で，その名のとおり高速に読み取れるという特徴がある．従来，おもにFAや納品管理などの分野で使用されてきたが，ディスプレイ・メディアとの相性が良いことから，最近になってQRコードの読み取り機能が付いた携帯電話も普及してきた．携帯電話を用いたチケット認証などの用途で使用されるケースもあり，今後，さまざまな分野での応用が見込まれる技術である．



図1
二次元コード「QRコード」

本稿では，QRコードのエンコード方法について解説する．QRコードにはオリジナルの「モデル1」と拡張型の「モデル2」の二つのモデルがあるが，今回は現在よく使用されているモデル2について取り上げる．

● QRコードの構造と用語

QRコードは白色または黒色の小さな正方形によって構成されている．この小さい正方形の1マスを「モジュール」または「セル」と呼ぶ．また，QRコードの大きさをバージョン(型番)と呼ぶ．QRコード・モデル2では，バージョン1から40まで存在し，バージョン1は21×21のモジュールで構成される．バージョンが一つ上がるごとに1辺あたり4モジュールずつ増加し，バージョン40では177×177モジュールとなっている．

QRコードは，リード・ソロモン誤り訂正符号により，汚れや破れ，欠けなど(データ誤り)があっても，それがある一定量以内であればデータを復元できる．この復元レベルについて，四つのレベルが設定できる．これを「誤り訂正レベル」と呼び，表1の4種類から選択できる．同じバージョンなら誤り訂正レベルが高いほどデータ容量が少なくなり，逆に同じデータ量で誤り訂正レベルを高くしようとすると大きなバージョンが必要になる．

表1 誤り訂正レベル

誤り訂正レベル	誤り訂正レベル指示子	復元能力(概数)
L	01	7%
M	00	15%
Q	11	25%
H	10	30%

注1：現在，QRコードは株式会社デンソーより分社した株式会社デンソーウェーブの登録商標である．

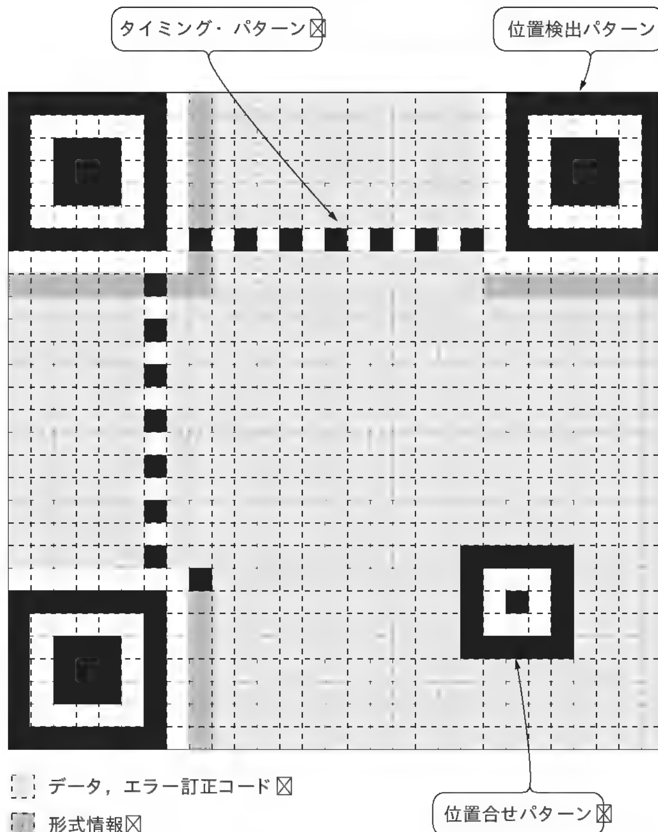


図2　バージョン3の構造図

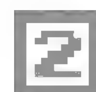
表2　各バージョンおよびエラー訂正レベルでのデータ容量

バージョン	誤り訂正レベル	データ・コード語数	誤り訂正コード語数	数字	英数字	8ビット	漢字
1	L	19	7	41	25	17	10
	M	16	10	34	20	14	8
	Q	13	13	27	16	11	7
	H	9	17	17	10	7	4
2	L	34	10	77	47	32	20
	M	28	16	63	38	26	16
	Q	22	22	48	29	20	12
	H	16	28	34	20	14	8
3	L	55	15	127	77	53	32
	M	44	26	101	61	42	26
	Q	34	36	77	47	32	20
	H	26	44	58	35	24	15
4	L	80	20	187	114	78	48
	M	64	36	149	90	62	38
	Q	48	52	111	67	46	28
	H	36	64	82	50	34	21
5	L	108	26	255	154	106	65
	M	86	48	202	122	84	52
	Q	62	72	144	87	60	37
	H	46	88	106	64	44	27
6	L	136	36	322	195	134	82
	M	108	64	255	154	106	65
	Q	76	96	175	108	74	45
	H	60	112	139	84	58	36
7	L	156	40	370	224	154	95
	M	124	72	293	178	122	75
	Q	88	108	207	125	86	53
	H	66	130	154	93	64	39
8	L	194	48	461	279	192	118
	M	154	88	365	221	152	93
	Q	110	132	259	157	108	66
	H	86	156	202	122	84	52
9	L	232	60	552	335	230	141
	M	182	110	432	262	180	111
	Q	132	160	312	189	130	80
	H	100	192	235	143	98	60
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

図2はバージョン3の構造図である．バージョン3は29×29モジュールで構成される．この中に読み取り時に位置検出を助ける「位置検出パターン」，画像のゆがみ補正などに使用される「位置合せパターン」，モジュールの座標を決定する「タイミング・パターン」が配置される．これらは「機能パターン」と呼ばれ，各バージョンごとに決まった位置に決まった白黒配置を行う．「形式情報」は誤り訂正レベルとマスク・パターン（後述）の情報が格納される．それ以外の部分がデータ部分（データ・コード語と誤り訂正コード語）となる．なおバージョン7以降では「型番情報」の領域があり，その部分にバージョンの情報が格納される．また実際にQRコードを配置する際は，QRコードの周りに「クワイエット・ゾーン」と呼ばれる4モジュール分以上の空白をあけておく必要がある．

● データのエンコード・モード

QRコードは内部のエンコード・モードとして数字，英数字，8ビット・バイト，漢字などの各モードがあり，これらを複合して持つこともできる．数字モードはその名のとおり0から9の数字だけのデータ列に対し3文字を10ビットで，英数字モードは数字およびアルファベットの大文字に加え空白と$%*＋－．/:の各記号を含むデータ列に対し2文字を11ビットで，8ビット・バイト・モードはバイナリ・データを含めすべてのデータ列に対し1バイトを8ビットで，漢字モードはシフトJISコードに基づいたデータ列に対し1文字を13ビットでそれぞれ内部エンコードを行う．各バージョン，誤り訂正レベルでのデータ容量を表2に示す．



QRコードの作成



ここでは実際に「http://www.cqpub.co.jp/interface/」というデータ列を例に，QRコードのエンコード手順（作成手順）を解説する．

● バージョンと誤り訂正レベルの決定

まずQRコードのバージョンと誤り訂正レベルを決定する．現実にはQRコードを印刷できる領域は限られている場合が多

表3　RSブロック

バージョン	誤り訂正レベル	RSブロック数	RSブロック
1	L	1	(26, 19, 2)
	M	1	(26, 16, 4)
	Q	1	(26, 13, 6)
	H	1	(26, 9, 8)
2	L	1	(44, 34, 4)
	M	1	(44, 28, 8)
	Q	1	(44, 22, 11)
	H	1	(44, 16, 14)
3	L	1	(70, 55, 7)
	M	1	(70, 44, 13)
	Q	2	(35, 17, 9)
	H	2	(35, 13, 11)
4	L	1	(100, 80, 10)
	M	2	(50, 32, 9)
	Q	2	(50, 24, 13)
	H	4	(25, 9, 8)
5	L	1	(134, 108, 13)
	M	2	(67, 43, 12)
	Q	2 2	(33, 15, 9) (34, 16, 9)
	H	2 2	(33, 11, 11) (34, 12, 11)
6	L	2	(86, 68, 9)
	M	4	(43, 27, 8)
	Q	4	(43, 19, 12)
	H	4	(43, 15, 14)
7	L	2	(98, 78, 10)
	M	4	(49, 31, 9)
	Q	2 4	(32, 14, 9) (33, 15, 9)
	H	4 1	(39, 13, 13) (40, 14, 13)
8	L	2	(121, 97, 12)
	M	2 2	(60, 38, 11) (61, 39, 11)
	Q	4 2	(40, 18, 11) (41, 19, 11)
	H	4 2	(40, 14, 13) (41, 15, 13)
9	L	2	(146, 116, 15)
	M	3 2	(58, 36, 11) (59, 37, 11)
	Q	4 4	(36, 16, 10) (37, 17, 10)
	H	4 4	(36, 12, 12) (37, 13, 12)
⋮	⋮	⋮	⋮

RSブロックはかっこ内順に(総コード語数，データ・コード語数，誤り訂正数)を意味する．
たとえば，バージョン5-Qならばデータ・コード語列を，33，33，34，34の四つに分割する．

く，その範囲の中でバージョンを大きくしようとすると，結果として1モジュールの大きさが小さくなり，コード・リーダのスペック(CCDや光学性能)から外れてしまうことも考えられる．また，携帯電話のように読み取れるバージョンが制限されている場合もある．データ量と出力範囲，コード・リーダの性能などを総合的に勘案し，バージョンと誤り訂正レベルを決定する．

今回は，かりにバージョン3を使用するものとする．データに英小文字があることから8ビット・バイト・モードとなり，33文字あるので表2から誤り訂正レベルはMとなる．

● データ・コード語の作成

まず最初にモード指示子を4ビットで指定する．8ビット・バイト・モードは0100である．

次に，文字数指示子を指定する．8ビット・バイト・モードのバージョン1～9では文字数を8ビットで表す．データは33文字あるので00100001となる．

続いて，実データ列を2進化する．

8ビット・バイト・モードの場合，そのまま文字コードを2進数化したものが該当する．例の場合，途中を省略するが，

h　t　t　p　/

01101000, 01110100, 01110100, 01110000, …, 00101111

のようになる．

最後に終端パターンを付ける．これは4ビットで0000である．なお，データ容量が実データのエンコードですでに満たされている場合は，これを省略できる．残りが4ビット未満の場合は短縮して付加する．

ここまで作成したビット列は，

0100, 00100001, 01101000, …, 00101111, 0000

となっているが，これを8ビットごとに区切る．もし最後が8ビット未満になれば0を付加し，8ビット単位にする．

01000010, 00010110, 10000111, …, 11110000

この8ビットごとのデータを「コード語」と呼ぶ．

ここでコード語数が表2のデータ・コード語の容量を満たしているかどうかを確認する．満たしていなければ「埋め草コード」と呼ばれる11101100および00010001を交互に付加して容量を埋める．現在，データ・コード語数が35なので，容量である44になるまでこれらのコードを付加する．

2進数のままでは見にくいので，10進数で表記すると，

66, 22, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114, 230, 55, 23, 7, 86, 34, 230, 54, 242, 230, 167, 2, 246, 150, 231, 70, 87, 38, 102, 22, 54, 82, 240, 236, 17, 236, 17, 236, 17, 236, 17, 236

となる．

● 誤り訂正コード語の作成

前述のように，QRコードはデータ・コード語にリード・ソロモン誤り訂正符号による冗長コード語を付加することにより，一定量のデータの欠損や誤りから元のデータを復元できるようになっている．次にQRコードを作成するうえでもっとも難解と思われる誤り訂正コード語の作成方法について説明する．

最初に，先に得たコード語列を各バージョンと誤り訂正レベルで規定されるRSブロック(表3)に分割する．たとえば，バージョン3-MはRSブロック数が1なので分割不要だが，これが

2以上の場合は分割し，各ブロックごとに以下の処理を行う．

JIS X0510(1999)では「QRコードの多項式は2を法とする算術および100011101を法とする算術(体の原始多項式 $x^8+x^4+x^3+x^2+1$ の係数を示す100011101を持つ 2^8 のガロア体)を使用して計算する．データ・コード語は多項式の項の係数で最高次項を最初のデータ・コード語とする．(中略)誤り訂正コード語はRS誤り訂正で使用される多項式 $g(x)$ によってデータ・コード語を除算して得られた剰余とする」とある．この記述を理解するためには「ガロア体とは？」といった情報数学を正しく理解する必要があるが，ここではそれは省略し，計算方法だけを説明する．

まず，先のJIS中の記述で出てくる $g(x)$ だが，これは表4を参照されたい．

バージョン3-Mは誤り訂正コード数が26なので，

$$g(x)=x^{26}+\alpha^{173}x^{25}+\alpha^{125}x^{24}+\alpha^{158}x^{23}+\alpha^{2}x^{22}+\alpha^{103}x^{21}+\alpha^{182}x^{20}$$
$$+\alpha^{118}x^{19}+\alpha^{17}x^{18}+\alpha^{145}x^{17}+\alpha^{201}x^{16}+\alpha^{111}x^{15}+\alpha^{28}x^{14}+\alpha^{165}x^{13}$$
$$+\alpha^{53}x^{12}+\alpha^{161}x^{11}+\alpha^{21}x^{10}+\alpha^{245}x^{9}+\alpha^{142}x^{8}+\alpha^{13}x^{7}+\alpha^{102}x^{6}$$
$$+\alpha^{48}x^{5}+\alpha^{227}x^{4}+\alpha^{153}x^{3}+\alpha^{145}x^{2}+\alpha^{218}x+\alpha^{70}$$

を使用する．ここで出てくる α とは「GF(2^8)上の原始要素2の根」とJISにあるが，ここではその特徴だけをあげておく．

1. 四則演算が行える
2. $\alpha^{255}=1$ である
3. α のべき乗と整数とをテーブル(表5)を用いてたがいに変換できる

次にデータ・コード語を係数とした多項式 $f(x)$ を $g(x)$ で除算し，剰余多項式を求める．ここで $f(x)$ は最低，次が $g(x)$ の最高次と同じ次数になるようにしておく．

$$f(x)=66x^{69}+22x^{68}+135x^{67}+71x^{66}+71x^{65}+3x^{64}+162x^{63}+242x^{62}$$
$$+247x^{61}+119x^{60}+119x^{59}+114x^{58}+230x^{57}+55x^{56}+23x^{55}+7x^{54}$$
$$+86x^{53}+34x^{52}+230x^{51}+54x^{50}+242x^{49}+230x^{48}+167x^{47}+2x^{46}$$
$$+246x^{45}+150x^{44}+231x^{43}+70x^{42}+87x^{41}+38x^{40}+102x^{39}+22x^{38}$$
$$+54x^{37}+82x^{36}+240x^{35}+236x^{34}+17x^{33}+236x^{32}+17x^{31}+236x^{30}$$
$$+17x^{29}+236x^{28}+17x^{27}+236x^{26}$$

これを $g(x)$ で除算する．

ここで $f(x)$ の最初の項の係数が66なので商の最初の項の係数は66となるが，$g(x)$ の係数が α のべき乗表記なので表5より66の α のべき乗表記，すなわち α^{139} を係数とする．

$g(x)\times(\alpha^{139})\times x^{43}=g(x)'$ とすると，

$$g(x)'=\alpha^{139}\times x^{69}+\alpha^{139}\times\alpha^{173}x^{68}+\alpha^{139}\times\alpha^{125}x^{67}+\cdots$$
$$=\alpha^{139}x^{69}+\alpha^{312}x^{68}+\alpha^{264}x^{67}+\cdots \quad (べき乗の計算)$$
$$=\alpha^{139}x^{69}+\alpha^{57}x^{68}+\alpha^{9}x^{67}+\cdots \quad (\alpha^{255}=1から)$$
$$=66x^{69}+186x^{68}+58x^{67}+\cdots \quad (整数表記)$$

$g(x)'$ と $f(x)$ との係数の排他論理和を取ると以下が求まる．

$$f(x)'=172x^{68}+189x^{67}+\cdots$$

以下，同様に $172=\alpha^{220}$ を利用し，$g(x)\times(\alpha^{220})\times x^{42}$ を計算し，$f(x)'$ との排他論理和をとる．あとはこれを繰り返し，最終的な剰余を求め，その係数を抜き出す．

表4　RS誤り訂正で使用される多項式 $g(x)$

誤り訂正コード語数	生成多項式 $g(x)$
7	$x^7+\alpha^{87}x^6+\alpha^{229}x^5+\alpha^{146}x^4+\alpha^{149}x^3+\alpha^{238}x^2+\alpha^{102}x+\alpha^{21}$
10	$x^{10}+\alpha^{251}x^9+\alpha^{67}x^8+\alpha^{46}x^7+\alpha^{61}x^6+\alpha^{118}x^5+\alpha^{70}x^4+\alpha^{64}x^3+\alpha^{94}x^2+\alpha^{32}x+\alpha^{45}$
13	$x^{13}+\alpha^{74}x^{12}+\alpha^{152}x^{11}+\alpha^{176}x^{10}+\alpha^{100}x^9+\alpha^{86}x^8+\alpha^{100}x^7+\alpha^{106}x^6+\alpha^{104}x^5$ $+\alpha^{130}x^4+\alpha^{218}x^3+\alpha^{206}x^2+\alpha^{140}x+\alpha^{78}$
15	$x^{15}+\alpha^{8}x^{14}+\alpha^{183}x^{13}+\alpha^{61}x^{12}+\alpha^{91}x^{11}+\alpha^{202}x^{10}+\alpha^{37}x^9+\alpha^{51}x^8+\alpha^{58}x^7+\alpha^{58}x^6$ $+\alpha^{237}x^5+\alpha^{140}x^4+\alpha^{124}x^3+\alpha^{5}x^2+\alpha^{99}x+\alpha^{105}$
16	$x^{16}+\alpha^{120}x^{15}+\alpha^{104}x^{14}+\alpha^{107}x^{13}+\alpha^{109}x^{12}+\alpha^{102}x^{11}+\alpha^{161}x^{10}+\alpha^{76}x^9+\alpha^{3}x^8$ $+\alpha^{91}x^7+\alpha^{191}x^6+\alpha^{147}x^5+\alpha^{169}x^4+\alpha^{182}x^3+\alpha^{194}x^2+\alpha^{225}x+\alpha^{120}$
17	$x^{17}+\alpha^{43}x^{16}+\alpha^{139}x^{15}+\alpha^{206}x^{14}+\alpha^{78}x^{13}+\alpha^{43}x^{12}+\alpha^{239}x^{11}+\alpha^{123}x^{10}+\alpha^{206}x^9$ $+\alpha^{214}x^8+\alpha^{147}x^7+\alpha^{24}x^6+\alpha^{99}x^5+\alpha^{150}x^4+\alpha^{39}x^3+\alpha^{243}x^2+\alpha^{163}x+\alpha^{136}$
18	$x^{18}+\alpha^{215}x^{17}+\alpha^{234}x^{16}+\alpha^{158}x^{15}+\alpha^{94}x^{14}+\alpha^{184}x^{13}+\alpha^{97}x^{12}+\alpha^{118}x^{11}+\alpha^{170}x^{10}$ $+\alpha^{79}x^9+\alpha^{187}x^8+\alpha^{152}x^7+\alpha^{148}x^6+\alpha^{252}x^5+\alpha^{179}x^4+\alpha^{5}x^3+\alpha^{98}x^2+\alpha^{96}x+\alpha^{153}$
20	$x^{20}+\alpha^{17}x^{19}+\alpha^{60}x^{18}+\alpha^{79}x^{17}+\alpha^{50}x^{16}+\alpha^{61}x^{15}+\alpha^{163}x^{14}+\alpha^{26}x^{13}+\alpha^{187}x^{12}$ $+\alpha^{202}x^{11}+\alpha^{180}x^{10}+\alpha^{221}x^9+\alpha^{225}x^8+\alpha^{83}x^7+\alpha^{239}x^6+\alpha^{156}x^5+\alpha^{164}x^4+\alpha^{212}x^3$ $+\alpha^{212}x^2+\alpha^{188}x+\alpha^{190}$
22	$x^{22}+\alpha^{210}x^{21}+\alpha^{171}x^{20}+\alpha^{247}x^{19}+\alpha^{242}x^{18}+\alpha^{93}x^{17}+\alpha^{230}x^{16}+\alpha^{14}x^{15}+\alpha^{109}x^{14}$ $+\alpha^{221}x^{13}+\alpha^{53}x^{12}+\alpha^{200}x^{11}+\alpha^{74}x^{10}+\alpha^{8}x^9+\alpha^{172}x^8+\alpha^{98}x^7+\alpha^{80}x^6+\alpha^{219}x^5$ $+\alpha^{134}x^4+\alpha^{160}x^3+\alpha^{105}x^2+\alpha^{165}x+\alpha^{231}$
23	$x^{24}+\alpha^{229}x^{23}+\alpha^{121}x^{22}+\alpha^{135}x^{21}+\alpha^{48}x^{20}+\alpha^{211}x^{19}+\alpha^{117}x^{18}+\alpha^{251}x^{17}+\alpha^{126}x^{16}$ $+\alpha^{159}x^{15}+\alpha^{180}x^{14}+\alpha^{169}x^{13}+\alpha^{152}x^{12}+\alpha^{192}x^{11}+\alpha^{226}x^{10}+\alpha^{228}x^9+\alpha^{218}x^8$ $+\alpha^{111}x^7+x^6+\alpha^{117}x^5+\alpha^{232}x^4+\alpha^{87}x^3+\alpha^{96}x^2+\alpha^{227}x+\alpha^{21}$
26	$x^{26}+\alpha^{173}x^{25}+\alpha^{125}x^{24}+\alpha^{158}x^{23}+\alpha^{2}x^{22}+\alpha^{103}x^{21}+\alpha^{182}x^{20}+\alpha^{118}x^{19}+\alpha^{17}x^{18}$ $+\alpha^{145}x^{17}+\alpha^{201}x^{16}+\alpha^{111}x^{15}+\alpha^{28}x^{14}+\alpha^{165}x^{13}+\alpha^{53}x^{12}+\alpha^{161}x^{11}+\alpha^{21}x^{10}$ $+\alpha^{245}x^9+\alpha^{142}x^8+\alpha^{13}x^7+\alpha^{102}x^6+\alpha^{48}x^5+\alpha^{227}x^4+\alpha^{153}x^3+\alpha^{145}x^2+\alpha^{218}x+\alpha^{70}$
28	$x^{28}+\alpha^{168}x^{27}+\alpha^{223}x^{26}+\alpha^{200}x^{25}+\alpha^{104}x^{24}+\alpha^{224}x^{23}+\alpha^{234}x^{22}+\alpha^{108}x^{21}+\alpha^{180}x^{20}$ $+\alpha^{110}x^{19}+\alpha^{190}x^{18}+\alpha^{195}x^{17}+\alpha^{147}x^{16}+\alpha^{205}x^{15}+\alpha^{27}x^{14}+\alpha^{232}x^{13}+\alpha^{201}x^{12}$ $+\alpha^{21}x^{11}+\alpha^{43}x^{10}+\alpha^{245}x^9+\alpha^{87}x^8+\alpha^{42}x^7+\alpha^{195}x^6+\alpha^{212}x^5+\alpha^{119}x^4$ $+\alpha^{242}x^3+\alpha^{37}x^2+\alpha^{9}x+\alpha^{123}$
30	$x^{30}+\alpha^{41}x^{29}+\alpha^{173}x^{28}+\alpha^{145}x^{27}+\alpha^{152}x^{26}+\alpha^{216}x^{25}+\alpha^{31}x^{24}+\alpha^{179}x^{23}+\alpha^{182}x^{22}$ $+\alpha^{50}x^{21}+\alpha^{48}x^{20}+\alpha^{110}x^{19}+\alpha^{86}x^{18}+\alpha^{239}x^{17}+\alpha^{96}x^{16}+\alpha^{222}x^{15}+\alpha^{125}x^{14}+$ $\alpha^{42}x^{13}+\alpha^{173}x^{12}+\alpha^{226}x^{11}+\alpha^{193}x^{10}+\alpha^{224}x^9+\alpha^{130}x^8+\alpha^{156}x^7+\alpha^{37}x^6+\alpha^{251}x^5+$ $\alpha^{216}x^4+\alpha^{238}x^3+\alpha^{40}x^2+\alpha^{192}x+\alpha^{180}$

最終的な誤り訂正コード語は，

219, 216, 43, 38, 178, 53, 241, 197, 133, 89, 66, 230, 43, 213, 204, 56, 77, 89, 137, 148, 123, 84, 72, 241, 214, 239

となる．

よって，最終的なコード語列はデータ・コード語と誤り訂正コード語をあわせた，

66, 22, 135, 71, 71, 3, 162, 242, 247, 119, 119, 114, 230, 55, 23, 7, 86, 34, 230, 54, 242, 230, 167, 2, 246, 150, 231, 70, 87, 38, 102, 22, 54, 82, 240, 236, 17, 236, 17, 236, 17, 236, 17, 236, 219, 216, 43, 38, 178, 53, 241, 197, 133, 89, 66, 230, 43, 213, 204, 56, 77, 89, 137, 148, 123, 84, 72, 241, 214, 239

となる．

Perlでの実装例をリスト1，リスト2に示す．なお，これらのプログラムは，本誌のWebサイト(`http://www.cqpub.co.jp/interface/`)からダウンロードできる．

● データの配置

QRコードでは1モジュール1ビットを意味するので，先に

表5　GF(2^8)のαのべき乗と整数の対応表

αのべき乗	整数	整数	αのべき乗	αのべき乗	整数	整数	αのべき乗
0	1			36	37	36	225
1	2	1	0	37	74	37	36
2	4	2	1	38	148	38	15
3	8	3	25	39	53	39	33
4	16	4	2	40	106	40	53
5	32	5	50	41	212	41	147
6	64	6	26	42	181	42	142
7	128	7	198	43	119	43	218
8	29	8	3	44	238	44	240
9	58	9	223	45	193	45	18
10	116	10	51	46	159	46	130
11	232	11	238	47	35	47	69
12	205	12	27	48	70	48	29
13	135	13	104	49	140	49	181
14	19	14	199	50	5	50	194
15	38	15	75	51	10	51	125
16	76	16	4	52	20	52	106
17	152	17	100	53	40	53	39
18	45	18	224	54	80	54	249
19	90	19	14	55	160	55	185
20	180	20	52	56	93	56	201
21	117	21	141	57	186	57	154
22	234	22	239	58	105	58	9
23	201	23	129	59	210	59	120
24	143	24	28	60	185	60	77
25	3	25	193	61	111	61	228
26	6	26	105	62	222	62	114
27	12	27	248	63	161	63	166
28	24	28	200	64	95	64	6
29	48	29	8	65	190	65	191
30	96	30	76	66	97	66	139
31	192	31	113	67	194	67	98
32	157	32	5	68	153	68	102
33	39	33	138	69	47	69	221
34	78	34	101	70	94	70	48
35	156	35	47				

(以下，省略．これ以降は本誌のWebサイトに掲載する)

得られたコード語列を2進化し，それぞれ所定のモジュールに配置する．なお，位置検出パターンおよびタイミング・パターンは各バージョンで固定なので，あらかじめ配置しておく．

配置のルールは，以下のとおりである．

1. 一番左上を(0, 0)とし i 行 j 列(i, j)の座標系を考える．たとえばバージョン3では(0, 0)～(28, 28)である．
2. スタートは右下．サンプルでは(28, 28)からスタートなので，ここにデータ(0 or 1)を，上位ビットから配置する．
3. 上下の進行方向を決める．最初の進行方向は上．
4. モジュール幅二つ分を基準とする．
   ⇒幅二つ分のうち右側にいるとき：左に移動することを試みる．空いていればそこに移動し，次のデータを配置する．もし，固定のパターンなどで埋まっていたら，現在の進行方向(上または下)へ一つ移動し，データを配置する．
   ⇒幅二つ分のうち左側にいるとき：そこより現在の進行方向側に空きがあるか確認する．空きがあれば，現在のモジュールに上下方向でもっとも近く，かつ幅二つのうち右側に優先してデータを配置する．もし進行方向に空きがなければ，一つ左方向へ移動し，そこにデータを配置する．また，そのとき進行方向を今までの方向と逆にする．

この説明では少しわかりにくいので，図3もあわせて参照いただきたい．配置後の状態は図4のようになる．

なお，RSブロックが二つ以上ある場合は，1番目のブロックの1番目のコード語，2番目のブロックの1番目のコード語…といったように，交互に配置していく．

● マスク処理

データを配置した状態で一方の色のモジュールが極端に多かったり，位置検出パターンに類似した模様があると，読み込みに支障をきたす恐れがある．これを防ぐためにQRコードでは8種類のマスク・パターンを用意し，その中で最適なものを選ぶようにしている．マスクをかける範囲は位置検出パターンやタイミング・パターン，形式情報などの機能パターンを除くデータ部で，表6の条件に該当する座標のモジュールをビット

かりに"01234567 89ABCDEF GHIJ KLMN"というビット列があり(実際には0か1しかないが…)，6行4列のマトリクスに配置したケース．☒

D	C	B	A	
F	E	9	8	
H	G	7	5	
J	I	5	4	
L	K	3	2	
N	M	1	0	(↑最初の進行方向)☒

また同様のマトリクスで中央の4行2列がすでに固定パターン"*"で埋まっている場合，"01234567 89ABCDEF"というビット列を配置したケース．☒

9	8	7	6☒	
A	*	*	5☒	
B	*	*	4☒	
C	*	*	3☒	
D	*	*	2☒	
E	F	1	0	(↑最初の進行方向)☒

図3　データ配置の方法



図4
データ配置後の状態

リスト 1　エラー訂正語生成処理（rs_ecc.pl）

```
#
# QRコード model2
# エラー訂正コード語生成
#

sub rsecc{

    local ($filename,@codewords) = @_;

    require $filename;    # 各バージョン，エラー・レベル用ファイル
    &set_rsecc_data;      # エラー訂正コード作成用データ読み込み

# RSブロック分割処理

    $i=0;
    $cnt=0;
    $rsblock_number=0;
    $rsblock_name="rs0";

    while($i<$max_data_codewords){
        $$rsblock_name[$cnt]=$codewords[$i];
        $cnt++;
        if ($cnt>=$rsdc[$rsorder[$rsblock_number]]){
            $cnt=0;
            $rsblock_number++;
            $rsblock_name="rs".$rsblock_number;
        }
        $i++;
    }

# エラー訂正語生成

    $rsblock_number=0;
    while ($rsblock_number<=$#rsorder){
        $rsblock_codewords=$rsorder[$rsblock_number];
                                                # ブロック内の総コード数
        $rsblock_codewords=~s/¥n//g;
        $rsdcs=$rsdc[$rsblock_codewords];
                                             # ブロックのデータ・コード語数
        $rsecs=$rsblock_codewords-$rsdcs;
                                             # ブロックのエラー・コード語数
        $rsblock_name="rs".$rsblock_number;

        $i=0;
        while($i<$rsblock_codewords){
            $gx_dash[$i]=0;        # g(x)'の係数  整数表記用配列初期化
            $i++;
        }

        @temp=@$rsblock_name;
        while($rsdcs>0){
            if ($temp[0]!=0){
                $q=&gf256log($temp[0]);
                                        # 項の最初の係数をべき乗表記変換
                $i=0;
                while ($i<=$rsecs){          # g(x)'を求める

                    # 係数について積をとり，結果を整数表記に変換

                    $gx_dash[$i]=&gf256ilog(&gf256product
                                            ($ks[$i],$q));
                    $i++;
                }
                $i=0;
                while ($i<=$rsecs){            # f(x)とg(x)'の和
                    $temp[$i]=($temp[$i] ^ $gx_dash[$i]);
                    $i++;
                }
            } else {                    # f(x)の最初の項の係数が0の場合
                if (!$temp[$rsecs]){
                    $temp[$rsecs]=0;
                }
            }
            shift(@temp);
            $rsdcs--;
        }
        push(@codewords,@temp);
                                # データ・コード語とエラー訂正コード語の結合
        $rsblock_number++;
    }
 @codewords;
}
```

```
#
# GF2^8
#

# べき乗→整数表記変換

sub gf256ilog{

my
@gf256il=(1,2,4,8,16,32,64,128,29,58,116,232,205,135,19,38,76,15
2,45,90,180,117,234,201,143,3,6,12,24,48,96,192,157,39,78,156,37
,74,148,53,106,212,181,119,238,193,159,35,70,140,5,10,20,40,80,1
60,93,186,105,210,185,111,222,161,95,190,97,194,153,47,94,188,10
1,202,137,15,30,60,120,240,253,231,211,187,107,214,177,127,254,2
25,223,163,91,182,113,226,217,175,67,134,17,34,68,136,13,26,52,1
04,208,189,103,206,129,31,62,124,248,237,199,147,59,118,236,197,
151,51,102,204,133,23,46,92,184,109,218,169,79,158,33,66,132,21,
42,84,168,77,154,41,82,164,85,170,73,146,57,114,228,213,183,115,
230,209,191,99,198,145,63,126,252,229,215,179,123,246,241,255,22
7,219,171,75,150,49,98,196,149,55,110,220,165,87,174,65,130,25,5
0,100,200,141,7,14,28,56,112,224,221,167,83,166,81,162,89,178,12
1,242,249,239,195,155,43,86,172,69,138,9,18,36,72,144,61,122,244
,245,247,243,251,235,203,139,11,22,44,88,176,125,250,233,207,131
,27,54,108,216,173,71,142,1);

$gf256il[$_[0]];

}


# 整数→べき乗表記変換

sub gf256log{
my
@gf256l=(0,0,1,25,2,50,26,198,3,223,51,238,27,104,199,75,4,100,2
24,14,52,141,239,129,28,193,105,248,200,8,76,113,5,138,101,47,22
5,36,15,33,53,147,142,218,240,18,130,69,29,181,194,125,106,39,24
9,185,201,154,9,120,77,228,114,166,6,191,139,98,102,221,48,253,2
26,152,37,179,16,145,34,136,54,208,148,206,143,150,219,189,241,2
10,19,92,131,56,70,64,30,66,182,163,195,72,126,110,107,58,40,84,
250,133,186,61,202,94,155,159,10,21,121,43,78,212,229,172,115,24
3,167,87,7,112,192,247,140,128,99,13,103,74,222,237,49,197,254,2
4,227,165,153,119,38,184,180,124,17,68,146,217,35,32,137,46,55,6
3,209,91,149,188,207,205,144,135,151,178,220,252,190,97,242,86,2
11,171,20,42,93,158,132,60,57,83,71,109,65,162,31,45,67,216,183,
123,164,118,196,23,73,236,127,12,111,246,108,161,59,82,41,157,85
,170,251,96,134,177,187,204,62,90,203,89,95,176,156,169,160,81,1
1,245,22,235,122,117,44,215,79,174,213,233,230,231,173,232,116,1
4,244,234,168,80,88,175);

    $gf256l[$_[0]];
}


# べき乗表記の積の計算

sub gf256product{
    (($_[0] + $_[1]) % 255);
}

1;
```

リスト 2　バージョン 3-M 用エラー訂正語生成用データ（qrv3_0e.pl）

```
sub set_rsecc_data{
#/* --- version 3-M -- */

# データ・コード語数
$max_data_codewords=44;

# 1RSブロックにおけるデータ・コード語数
$rsdc[70]=44;

# g(x)の係数
@ks=(0,173,125,158,2,103,182,118,17,145,201,111,28,165,
53,161,21,245,142,13,102,48,227,153,145,218,70);

# RSブロックの各コード語数
@rsorder=(70);
}
1;
```

表6　マスク条件

マスク・パターン参照子	条件
000	$(i+j) \bmod 2 = 0$
001	$i \bmod 2 = 0$
010	$j \bmod 3 = 0$
011	$(i+j) \bmod 3 = 0$
100	$((i \text{ div } 2)+(j \text{ div } 3)) \bmod 2 = 0$
101	$(i \times j) \bmod 2 + (i \times j) \bmod 3 = 0$
110	$((i \times j) \bmod 2 + (i \times j) \bmod 3) \bmod 2 = 0$
111	$((i \times j) \bmod 3 + (i+j) \bmod 2) \bmod 2 = 0$

div ：整数の除算
mod：除算後の整数剰余

たとえば，マスク・パターン000では
(20, 20)の場合，(20+20) mod 2 = 0なので　ビットを反転させる．
(19, 20)の場合，(19+20) mod 2 = 1なのでビットを反転させない．

表7　マスク評価の条件

特　徴	評価条件	失点
同色の行/列の隣接ブロック	モジュール数=(5+i)	3+i
同色のモジュール・ブロック	ブロック・サイズ2×2	3
行/列における1:1:3:1:1(暗:明:暗:明:暗)のパターン		40
全体に占める暗モジュールの割合	50±(5+k)%～50±(5+(k+1))%	10×k

反転させる．この8種類のマスク・パターンそれぞれについて表7の条件で評価し，もっとも失点の少ないパターンを選択する．例では011を選択した．

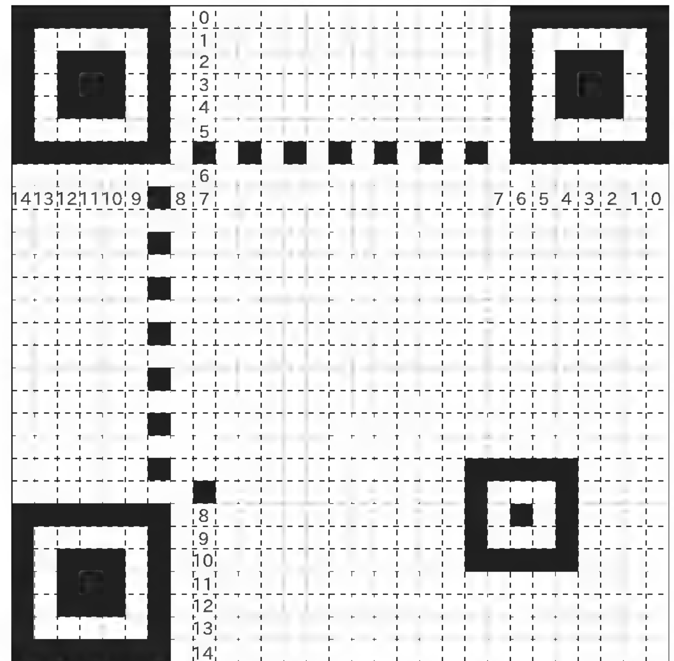
なお，マスクの選択を誤っても読み込みができなくなることはほとんどない．評価の基準に若干のあいまいさを感じるが，必要以上に気にすることはないだろう．

● 形式情報の算出と配置

誤り訂正レベルとマスク・パターン参照子を記述するため，15ビットの形式情報を作成する．最初の2ビットは表1の誤り訂正レベル指示子に対応する数字を当てはめ，次の3ビットは前項で選択したマスク参照子を割り当てる．残りの10ビットは誤り訂正ビットとして用いる．誤り訂正符号にはBose-Chaudhuri-Hocquenghem(以下BCH)の(15, 5)符号を使用する．先に得られた5ビットに対し，各ビットが係数となり，最下位ビットがx^{10}の係数となる多項式を考える．例の場合，00 011なので，$x^{11}+x^{10}$である．

これを$G(x)=x^{10}+x^8+x^5+x^4+x^2+x+1$で割った剰余を求める．これを計算すると剰余$R(x)=x^9+x^8+x^6+x^4+x^3+1$が求められる．よって，ビット列は00 011 1101011001となる．

次に，できたビット列がすべて0にならないように，101010000010010のビット列と排他論理和をとる．最終的なビット列は101101101001011になる．



上位ビットを14から配置する．

図5　形式情報の配置

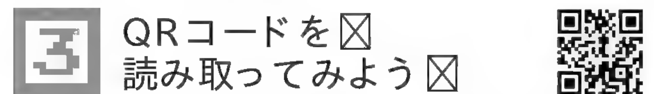


図6
完成したQRコード

なお，実際にエンコードする際は，これらの組み合わせはたかだか2^5=32通りしかないので，あらかじめ計算したものを使用してもよいだろう．

最後にこのビット列を図5の位置に配置する．

これで完成である(図6)．



3 QRコードを読み取ってみよう

それでは，先ほど作ったQRコードを読み取ってみよう．

QRコードを読み取るためのスキャナやプログラムも発売されているが，ここでは手軽にQRコードに対応している携帯電話から読み込んでみよう(写真1)．

メニューから「バーコード読み取り」を選択し，カメラをマクロ・モードに切り替える．

画面にQRコードがおさまるようにして，読み取りボタンを



用途にあわせて選べる

QRコード対応のスキャナ

編集部

QRコードを用いたシステムを構築する場合には，QRコードを印字するプリンタと，QRコードを読み取るためのスキャナが必要となる．本稿では，QRコードの読み取りの際にQRコードに対応した携帯電話を使ったが，用途にあわせた専用のスキャナ注Aも多数，発売されている．

ここでは，QRコード用のスキャナについて，QRコードを開発したデンソーウェーブの製品注Bを例に，現在，どのような製品があるのかを見ていく．

● コンピュータと組み合わせて使うタイプ（写真A）

ディスプレイはついておらず，パソコンなどのコンピュータと組み合わせて使うタイプ．専用のソフトウェアでコンピュータからパラメータの設定などが行える．工場での入出荷管理や伝票管理などの用途で使用できる．

● 持ち運びに便利なタイプ（写真B）

ディスプレイが付いており，読み取ったQRコードの内容がその場で確認できる持ち運びに便利なタイプ．屋外での施設への入場管理や棚卸，在庫管理など，さまざまな用途で使用できる．

● 携帯電話の液晶画面に対応したタイプ（写真C）

携帯電話の液晶画面に表示されたQRコードを読み取るための据え置き型のスキャナ．携帯電話を用いたチケット認証や，クーポン券の確認などの用途で使用できる．

● そのほか

そのほかにも，商品カタログや伝票などに掲載されている小さなQRコードを読み取るためのペン・タイプのものや，工場のベルト・コンベア上を流れている品物に付いているQRコードを読み取るための固定カメラ・タイプのスキャナなどもある．

注A：スキャナによっては，別途，デコード用のソフトウェアを用意する必要がある．

注B：デンソーウェーブから発売されているスキャナには，デコーダが組み込まれている．



写真A　コンピュータと組み合わせて使うタイプ（GT10Qシリーズ）

正しくスキャンできたことがわかるように，バイブレーション機能を付けるなどのくふうもなされている．連結したQRコードも読み取れる



写真B　持ち運びに便利なタイプ（BHT-300Qシリーズ）

QRコード以外の二次元コードやバーコードにも対応．無線LAN機能を使ったデータ転送もできる



写真C　携帯電話の液晶画面に対応したタイプ（QK11）

携帯電話のバックライトのON，OFFに関わらず読み取れる．雑誌から切り抜いたQRコードも読み取れる

押す．うまく読み込めていれば画面にデコードされたデータが出ているはずである注2．

今回の例のように，URLのデータであれば，そのままサイトに接続できる（今回の例はPC向けのサイトなので，携帯電話には向かないが…）．また，各キャリアが規定したフォーマットに基づいてエンコードすれば電話帳への入力も容易に行える．たとえば，名刺にQRコードを印刷しておけば，便利かもしれない．

機種によっては携帯アプリとの連携も可能だ．みなさんもいろいろな応用例を考えてみてはいかがだろうか．

すえたけ・よういち　http://www.swetake.com/

注2：携帯電話の機種によっては，操作方法が異なる．



写真1　QRコード対応のカメラ付き携帯電話でもQRコードを読み取ることができる

LinkerとLoaderの概念からテクニックまで

リンカを100%使いこなそう!

第1回 リンカとオブジェクト

坂井 弘亮

プログラムを書いてコンパイルした後，実行できるようになるまでの最終段階として「リンク」という過程があります．また，リンク後の実行形式をCPUが実行するためには，「ロード」という作業が必要です．リンク，ロードを行うアプリケーションを一般に「リンカ」，「ローダ」と呼びます．

コンパイル型のプログラミング言語では，ソース・コードをコンパイルした後に，複数のライブラリとリンクすることで，実行形式を作成します．プログラムを実行できるようにするためには，リンクは必須の過程です．また，実行形式は通常はファイル・システム上に格納されていますが，CPUはメイン・メモリ上の実行コードしか実行できません．実行形式のプログラムを実行するためには，ファイル・システムからメイン・メモリへのロードが必須になります．しかし，汎用OSの上でアプリケーションを作成・利用しているだけならば，これらは暗黙のうちに行われてしまうので，あまり意識しなくとも，とくに問題はありません．

アプリケーション・プログラムならばそれでもよいのですが，組み込み系のプログラムやOSの作成/移植の際には，そうはいきません．これらの場合には，オブジェクトの配置を自由に行いたいために，リンカの知識やカスタマイズ，細かいチューニングが必須になってくるからです．

本連載では，まず第1回で，リンカの一般的な動作について説明します．その次に，実験を通してさらに理解を深めます．たんなる確認実験ではなく，リンカの機能を利用しないと実現できない(C言語の文法の範囲では実現できない)ようなテクニックを紹介します．最後に，実際にどのようなところで利用されているのか，どのようなことに利用できるのかを説明します．

なお，とくに断りのない限りは，FreeBSD-4.10を題材としています．コンパイラ・リンカなどには，gccとbinutilsを利用しています．バージョンは，FreeBSD付属のgcc-2.95.4とbinutils-2.12.1です．また，一部ではNetBSDを参考にしていますが，これは原稿執筆時点(2004年10月3日)でのNetBSD-currentを利用しています．一部，Linuxカーネルについて言及している部分もありますが，これは本稿執筆時点での最新バージョンであるLinux-2.6.8.1を参照し，おもにLinux/PowerPCを題材にしています．

リンカとオブジェクト

組み込みシステムやOSを作るうえでは，リンカはメモリ配置と密接な関係がある部分になるので，リンカの知識が必須になります．しかし，リンカの動作や詳細について細かく説明してある資料は，書籍，雑誌記事，インターネットとも，あまりないように感じます．リンカの知識がないと，カーネルがメモリ上にどのようにマッピングされるか，メモリがどのように利用されるかといったイメージをつかみにくくなります．このため，OSのカーネル・ソースを読んだり，OSの移植に挑戦する際に，リンクの知識の有無が，壁の一つとなることが多々あるようです．

リンクについては，書籍では，次のように説明されていることが多いように思います．

「複数のオブジェクトどうしを結合することである」

しかし，初心者のうちは単一ファイルのプログラム(場合によっては`main()`しかないような)しか作成したことがなかったりして，どうもこの説明だと実感が湧かないことが多いようです(筆者自身がそうだった)．

第1回では，リンカが何を行っているのか，プログラムがコンパイルされ，実行形式となり，実際に動作するためには，何が必要なのかを説明します．

リンカの仕事

● リンカの必要性

何十万行もあるような，規模の大きなアプリケーションを作成することを想像してください．このようなとき，ソース・コードを単一のファイルにすべて押し込むようなことはまずありません．必ず複数のファイルに分割します．そうしないと，複数人で役割分担して同時開発することが，現実的に不可能になってしまうからです．

また，大規模なアプリケーションの場合には，すべてをコンパイルするまでに数時間かかってしまうことも珍しいことではありません．ソース・コードを複数のファイルに分割し，ファ

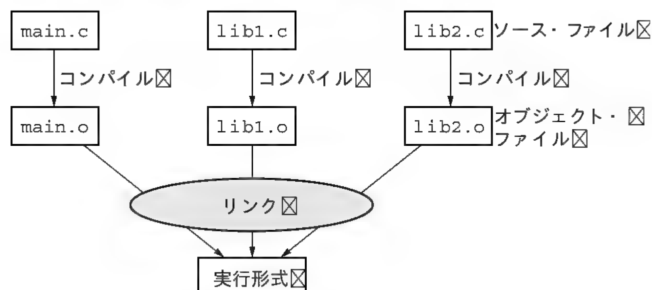


図1　コンパイルとリンク

イル単位でコンパイルを行うことで，ソース・コードの修正時には，必要なファイルのみコンパイルを行えばいいようにすることができます．

● ファイルを分割した時

ソース・コードを複数のファイルに分割した際，通常のコンパイラは，ファイル単位でコンパイルをすることができます[注1]．このときには「オブジェクト・ファイル」と呼ばれる一次ファイルが作成されます．オブジェクト・ファイルの拡張子は，通常は.oになります．オブジェクト・ファイルは，C言語のソース・コードを，ファイル単位で機械語に変換したものです．つまり，コンパイラは.cファイルに1対1に対応した.oファイルを作成することになります．

ファイルを分割すると，「あるファイルに実体が定義してある関数を，別のファイルから呼び出したい」という必要性が出てきます．つまり，ファイル間をまたいだ関数呼び出しが発生してきます．また，変数についても同様に，ファイルをまたいだ変数の参照や書き込みが発生します．このような場合，呼び出し元のファイルを単体でコンパイルするときには，呼び出し先の関数の実体が見当たらないので，完全な機械語コードを作成することができません．このためオブジェクト・ファイルでは，関数呼び出しの部分に，「ここでは別ファイルの×××という関数を呼び出す」というマークが残されます．最終的に複数のオブジェクト・ファイルを結合したときに，マークの部分は実際の関数呼び出しに置き換えられます．この結合作業が「リンク」です．リンクすることで，実際に実行可能なファイルができあがります．この実行可能なファイルのことを，「実行形式」と呼びます（図1）．なお，分割コンパイルの具体的な方法に関しては本稿の範囲を越えてしまうので，参考文献（1）を参照してください．

● 単一のファイルでもリンカは必要

では，単一のファイルからなるプログラムだったとしたら，リンクは必要ないのでしょうか？　たとえばhello.c（リスト1）を見てください．これはいわゆる“Hello world”ですが，このようなmain()関数しかないプログラムの場合はどうでしょうか．

リスト1　hello.c

```
#include <stdio.h>

int main()
{
  printf("Hello world!¥n");
  exit (0);
}
```

答は「それでもリンクは必要」です．単一のファイルからなるプログラムで，main()関数しかないようなプログラムだったとしても，内部ではライブラリ関数を利用しているかもしれません．hello.cでは，ライブラリ関数としてprintf()を利用しています．また，printf()だけでなくexit()も実はライブラリ関数なのです．このような場合には，ライブラリのリンクが必要になります．

● ライブラリとのリンク

また，一般的なプログラムであればシステム・コールを利用します．これらシステム・コールの呼び出しはアセンブラで書かれるのが普通です．アセンブリ言語で書かれたソース・ファイルを処理するのはアセンブラですが，C言語で書かれたソース・ファイルを処理するのはCコンパイラです．このため，アセンブリ言語で書かれたシステム・コール呼び出しをC言語側から利用するためには，アセンブリ言語のソースとC言語のソースを別々のオブジェクト・ファイルにして，最終的にリンクするような作業が必要です．このように，複数の種類の言語が混在する場合にも，リンクという作業が必要になります．

さらに，たとえライブラリ関数が利用されていないとしても，実行形式を作成する際には，スタートアップ・ルーチンという「初期化」を行うプログラムがリンクされます．

スタートアップ・ルーチンでは，レジスタの初期化やmain()への引き数（いわゆるargcとargv）の設定，スタック・ポインタの初期化などが行われます．実はプログラムを実行したときにいちばん最初に実行されるのはmain()ではありません．実際には，スタートアップ・ルーチンがいちばん最初に実行され，そこからmain()が呼び出されます．さらに言うならば，exit()をするとプログラムは即終了するわけではありません．exit()の後にいくつかの終了処理が行われた後，_exit()というシステム・コールによって，プロセスが終了します．

また，関数や変数の実際のアドレスへの配置も，リンクのときに行われます．このため，C言語（もしくはそれ以外のコンパイラ型言語）のプログラムを実行形式に変換するには，リンクという工程が絶対に必要なのです．

通常，コンパイラが行う作業は，C言語（もしくはそれ以外のコンパイラ型言語）のソース・コードを機械語に変換し，オブジェクト・ファイルを作成するだけです．リンカはこれらの機械語のコード（オブジェクト・ファイル）を結合し，実際に

注1：gccでは-cオプションをつけることで，ファイル単位にコンパイルを行うことができる．この場合には，単体のファイル中には，main()関数は必要ない．

リスト 2 リンクまでの流れ――hello.cをgcc -vでコンパイルしたときの出力結果

```
001: % gcc hello.c -Wall -o hello -v
002: Using builtin specs.
003: gcc version 2.95.4 20020320 [FreeBSD]
004:  /usr/libexec/cpp0 -lang-c -v -D__GNUC__=2 -D__GNUC_MINOR__=95 -Di386 -D__FreeBSD__=4 -D__FreeBSD_cc_version=460001 -Dunix
-D__i386__ -D__FreeBSD__=4 -D__FreeBSD_cc_version=460001 -D__unix__ -D__i386 -D__unix -Acpu(i386) -Amachine(i386) -Asystem(unix)
              -Asystem(FreeBSD) -Wall -Acpu(i386) -Amachine(i386) -Di386 -D__i386 -D__i386__ -D__ELF__ hello.c /tmp/ccqnUJxp.i
005: GNU CPP version 2.95.4 20020320 [FreeBSD] (i386 FreeBSD/ELF)
006: #include "..." search starts here:
007: #include <...> search starts here:
008:  /usr/include
009:  /usr/include
010: End of search list.
011: The following default directories have been omitted from the search path:
012:  /usr/include/g++
013: End of omitted list.
014:  /usr/libexec/cc1 /tmp/ccqnUJxp.i -quiet -dumpbase hello.c -Wall -version -o /tmp/ccbkxy21.s
015: GNU C version 2.95.4 20020320 [FreeBSD] (i386-unknown-freebsd) compiled by GNU C version 2.95.4 20020320 [FreeBSD].
016:  /usr/libexec/elf/as -v -o /tmp/ccs4WTKL.o /tmp/ccbkxy21.s
017: GNU assembler version 2.12.1 [FreeBSD] 2002-07-20 (i386-obrien-freebsd5.0) using BFD version 2.12.1 [FreeBSD] 2002-07-20
018:  /usr/libexec/elf/ld -V -dynamic-linker /usr/libexec/ld-elf.so.1 -o hello /usr/lib/crt1.o /usr/lib/crti.o /usr/lib/crtbegin.o
                                              -L/usr/lib /tmp/ccs4WTKL.o -lgcc -lc -lgcc /usr/lib/crtend.o /usr/lib/crtn.o
019: GNU ld version 2.12.1 [FreeBSD] 2002-07-20
020:   Supported emulations:
021:    elf_i386
022: %
```

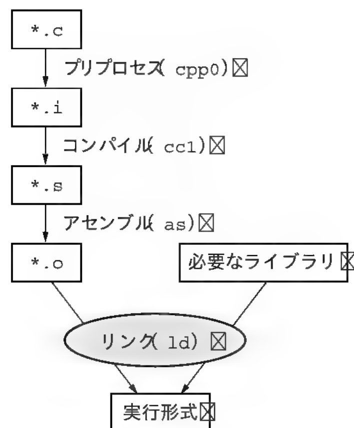


図2
gccが実際に行う作業

OSがロードして実行できるような実行形式に変換します．最終リンクの際には，ユーザが用意したオブジェクト・ファイルだけでなく，OSがシステムとして用意しているライブラリなどもリンクされます．つまりリンクとは，「コンパイラが出力した機械語のコードを，OS依存な実行形式に変換する段階」であるということができます．

● リンカの動作例

このように，プログラムを実行形式に変換するためには，リンクが必要です．では，先ほどのhello.cのようなプログラムをgccでコンパイルして実行形式を作成する際には，いつリンクが行われているのでしょうか．gccを-vオプションで実行すると，実行形式が作成されるまでの詳細が出力されます．リスト2は，hello.cをgcc -vでコンパイルしたときの出力結果です．

リスト2を見ると，18行目で，/usr/libexec/elf/ldというリンカが呼ばれていることがわかります．引き数には，crt1.o，crti.o，crtbegin.o，crtend.o，crtn.oの五つのオブジェクト・ファイルと，/tmp以下にある/tmp/ccs4WKTL.oというオブジェクト・ファイルが指定されています．前者の五つのファイルは，スタートアップ・ルーチンです(crtはC RunTime start upの略)．後者の/tmp/ccsWKTL.oは，hello.cから作成されたオブジェクト・ファイルです．また，4行目では/usr/libexec/cpp0(プリプロセッサ)が呼ばれ，16行目では/usr/libexec/elf/as(アセンブラ)が呼ばれています．

このように，リンクまでにはいくつかの作業が順番に行われます．/usr/libexec/elf/ldでのリンク時に/tmp/ccsWKTL.oという何やらよくわからない名前のオブジェクト・ファイルが指定されているのは，/tmpをテンポラリ・ディレクトリとして，前工程の出力(リスト2の場合には，16行目の/usr/libexec/elf/asの出力)を受け渡しているためです．

リスト2の14行目に注目してください．ここで/usr/libexec/cc1というコマンドが呼ばれています．gccは，指定されたファイルを読み込み，ファイルの形式に応じて，プリプロセス，コンパイル，アセンブル，リンクまでの作業を順番に行ってくれます．リンクも自動で行ってくれるため，プログラマはリンカを利用していることを意識せず，実行形式を作成することができます．

つまりgccは，プリプロセスからリンクまでの，実行形式作成用のマネージャとでもいうべきものであり，正しい意味での「コンパイラ」はcc1である，ということになります．このため，gccは「コンパイラ・ドライバ」と呼ばれることもあります(図2)．

コンパイルというのは，正確には，ソース・コードを機械語に変換する作業のことなので，リンクとはまったく別の作業です．しかし通常はコンパイルというと，上記のgccの動作のように，リンクまでの一連の作業を総称して「コンパイル」と呼ぶ

ことが多くあるようです．つまり，実行形式が作成されるまでを「コンパイル」と呼んでいるわけです．その意味では，cc1は狭義のコンパイラで，gccは広義のコンパイラだということができるでしょうか．

本稿では以後，これをはっきりと区別して，「コンパイル」は狭義の意味で使い，広義の「コンパイル」は「コンパイル・リンク」と呼びます[注2]．

関数や変数の実体は，特定のアドレス上に置かれます．プログラム中の関数呼び出しや変数の参照は，実際には，特定アドレスへのジャンプや，特定アドレスのメモリの参照になります．しかし，オブジェクト・ファイルの段階では，これらのアドレスは決定されていません（すべてのオブジェクト・ファイルが出そろった段階でないと，アドレスを決定できないため）．リンクは，これらの関数や変数を実際に特定アドレスに割り当てて，それらを利用している部分に，割り当てられたアドレスを挿入する作業であるともいえます．

セクション

● 実行形式のフォーマットについて

我々がアプリケーション・プログラムを作成し，gccによりコンパイル・リンクを行うと，実行形式が作成されます．この「実行形式」は，CPUが実行する機械語コードを「ベタに」ファイルにしたもの，というわけではありません．先頭部分にヘッダ情報を持ち，ある特定のフォーマットになっています．

このフォーマットには，古くはa.out（Assembler OUT）形式が利用されていましたが，現在ではELF（Executable and Linking Format）形式が多く利用されています．また一部では，COFF（Common Object File Format）という形式も利用されています．具体的なフォーマットに関しては，参考文献（2）を参照してください．また，これらのフォーマットを総称して，「オブジェクト・フォーマット」，「オブジェクト・ファイル・フォーマット」などと呼びます．呼び方は「オブジェクト・フォーマット」ですが，オブジェクト・ファイルに限らず，実行形式やダイナミック・リンク・ライブラリなども，このフォーマットで表されます．通常は*.oファイルのことを「オブジェクト・ファイル」と呼びますが，広義では*.oファイルだけに限らず，このように実行形式などのことも含めて「オブジェクト」と呼ぶ場合もあります．

● ファイル内の複数の領域

このように実行形式にはヘッダ情報が添付されています．さらに，ファイルの内部はその目的ごとに，複数の領域に分けられています．これらの各領域のサイズは，sizeコマンドで確認することができます．たとえば，リスト1のhello.cをコンパイル・リンクして実行形式helloを作成し，helloに対して

注2：一般には「コンパイル・リンク」のことを「ビルド」と呼ぶこともある．

リスト3　size helloの結果

```
% gcc hello.c -o hello -Wall
% ./hello
Hello world!
% size hello
   text    data     bss     dec     hex filename
   1042     208      28    1278     4fe hello
%
```

sizeを実行すると，リスト3のようになります．

リスト3を見ると，helloという実行形式が，text，data，bssという三つの領域からできていることがわかります．これらはそれぞれ，テキスト領域，データ領域，BSS領域（BSS：Block Started by Symbol）と呼ばれます．各領域名の下に表示されている数値は，各領域のバイト・サイズです．なお，decは三つの領域の合計サイズ，hexはそれを16進表記にしたものです．

テキスト領域には，機械語の実行コードが置かれます．また，変更されることのないデータ（const定義されている変数や，文字列リテラルの本体など）も，通常はここに置かれます．メモリ保護のあるOSの場合には，テキスト領域はread onlyに設定することで，コードやデータの不正な変更を防止します．

データ領域には，初期値のある変数の本体が置かれます．書き込みが可能な変数は，この領域に置かれます．ただしauto変数（ローカル変数で，スタック上に置かれるもの）は，対象外です．つまりデータ領域に置かれるのは，以下の変数のうち，初期値の定義してあるものです．

- 外部に公開している変数（グローバル変数）
- 関数の外部で定義しているstatic変数（ファイル内で広域だが，ファイル外には公開しない変数）
- 関数の内部で定義しているstatic変数（関数にローカルだが，値が保存される変数）

BSS領域には，初期値が未定義の変数が配置されます．こちらもauto変数は，対象外になります．つまり，上記の変数のうち，初期値の定義していないものがBSS領域に置かれます．

初期値が未定義の場合には，実行形式中にデータとして値をもつ必要はありません．したがってBSS領域は，実行形式中では，サイズの情報だけで，実体はありません．プログラムの実行のためにOSが実行形式をロードしたときに，メモリ上に作成されます．

アプリケーション・プログラムの場合には，実行形式を実際にメモリ上に展開して実行を開始するのは，OSの仕事です．プログラムの実行を行うには，exec()ファミリのシステム・コールを利用します．このときOSは，機械語コードをメモリ上のどこに展開するか，どこから実行するか，といったことを知る必要があります．ヘッダにこれらの情報をもたせることで，OSはどのように展開・実行すればよいかという情報を得ることができます．

OSはプログラムの実行時には，実行形式のヘッダを参照し，

そこに格納された情報に従って，それぞれの領域をメモリ上に展開し，実行を開始します注3．このように，実行形式をロードしてメモリ上に展開し，実際に実行を開始するためのプログラムを「ローダ」と呼びます．

a.out 形式では，領域はテキスト，データ，BSSの3種類しかとることができませんでした．また，a.out 形式では，領域にいろいろな属性を持たせることができません．これでは柔軟性に欠けるため，ELF 形式では，任意数の「セクション」を確保して，さまざまな属性をもたせることができるようになっています．

objdumpを利用すると，セクション情報が得られます．リスト4は，helloに対してobjdumpを実行して，セクション情報を表示させたときの結果です．

リスト5では番号が0～19の，20個の区画が表示されています．これらの区画のことを「セクション」と呼びます．また，これとは別に，「セグメント」ということばもあります．これらの「領域」，「セグメント」，「セクション」ということばの使い分けは，オブジェクト・フォーマットによって異なるので注意が必要です．上記のobjdump -hで得られる区画情報は，実はELF 形式でいうところの「セクション」に相当するので，本稿ではセクションと呼んでいます．また先述したテキスト，データ，BSSの三つに関しては，「領域」と呼ぶことにします．

リスト4　objdump -h helloの結果

```
% objdump -h hello

hello:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .interp       00000019  080480f4  080480f4  000000f4  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000018  08048110  08048110  00000110  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .hash         00000054  08048128  08048128  00000128  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .dynsym       00000100  0804817c  0804817c  0000017c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .dynstr       0000009d  0804827c  0804827c  0000027c  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .rel.plt      00000018  0804831c  0804831c  0000031c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .init         0000000b  08048334  08048334  00000334  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  7 .plt          00000040  08048340  08048340  00000340  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  8 .text         00000178  08048380  08048380  00000380  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  9 .fini         00000006  080484f8  080484f8  000004f8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 10 .rodata       0000000f  080484fe  080484fe  000004fe  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 .data         0000000c  08049510  08049510  00000510  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 12 .eh_frame     00000004  0804951c  0804951c  0000051c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 13 .dynamic      00000098  08049520  08049520  00000520  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 14 .ctors        00000008  080495b8  080495b8  000005b8  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 15 .dtors        00000008  080495c0  080495c0  000005c0  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 16 .got          00000018  080495c8  080495c8  000005c8  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 17 .bss          0000001c  080495e0  080495e0  000005e0  2**2
                  ALLOC
 18 .comment      000000a0  00000000  00000000  000005e0  2**0
                  CONTENTS, READONLY
 19 .note         00000050  00000000  00000000  00000680  2**0
                  CONTENTS, READONLY
%
```

● VMAとLMAの対応

リスト4では，VMAとLMAという2種類のアドレスが表示されています．VMAはVirtual Memory Addressの略で，セクションをリンクするときにベースとなるアドレスです．関数や変数のアドレスは，VMAを基準として配置されます．また，LMAはLoad Memory Addressの略で，セクションを展開する先のアドレスのことです．

通常のアプリケーションではVMA＝LMAとなりますが，例外としてVMA≠LMAとなることもあります．代表的なのはOSのカーネルです．仮想メモリで動作するOSの場合には，最初は物理アドレスで動作して，後に自分自身を仮想アドレスにマップしなおして，途中から仮想アドレスで動作する，というのが普通です．このような場合には，カーネルの展開先は物理アドレスになりますが，カーネル中の関数や変数は，実際にOSが動作する論理アドレスをベースにしてリンクすることになりますから，VMA≠LMAとなります注4．

● セクションの実例

リスト4を見ると，helloという実行形式は，実際には20個のセクションからできていることがわかります．しかしリスト3では，表示されたのはtext，data，bssの三つの領域だけでした．これらの結果の違いは何でしょうか．それは，sizeコマンドのソースを見てみるとわかります．

which sizeによると，sizeの本体は，/usr/bin/sizeとなっているので，セオリどおりにソースを追いかけるならば，そのソースは/usr/src/usr.bin以下にあることになります．そこで，/usr/src/usr.bin/sizeを見てみましょう．ここにはsize.cというファイルがあり，size.cを見ると，リスト5のようなことを行っている部分があります．どうやら，ヘッダ部分の解析を行っているようです．

リスト5では，11行目でファイルをオープンして，15行目でファイルの先頭をstruct execという構造体に読み込んでいます．struct execは，/usr/include/sys/imgact_aout.hでリスト6のように定義されています．

つまり，ファイルの先頭16バイトには，マジック・ナンバ，テキスト領域のサイズ，データ領域のサイズ，BSS領域のサイズが，4バイトずつ利用して格納されており，sizeコマンド

注3：実際には仮想メモリ機構のデマンド・ローディングにより，要求された部分のみがそのつどメモリ上に展開されるのだが，ここでは本質ではないので深くは言及しない．

注4：このため，仮想アドレスのマップ前に関数呼び出しやグローバル変数の操作をする場合には，仮想アドレスを物理アドレスに計算しなおしてアクセスするか，相対アドレスでアクセスする必要がある．

リスト 5　/usr/src/usr.bin/size/size.c より抜粋

```
int
show(count, name)
        int count;
        char *name;
{
        static int first = 1;
        struct exec head;
        u_long total;
        int fd;

        if ((fd = open(name, O_RDONLY, 0)) < 0) {
                warn("%s", name);
                return (1);
        }
        if (read(fd, &head, sizeof(head)) !=
                    sizeof(head) || N_BADMAG(head)) {
                (void)close(fd);
                warnx("%s: not in a.out format", name);
                return (1);
        }
        (void)close(fd);

        if (first) {
                first = 0;
                (void)printf(
                        "text¥tdata¥tbss¥tdec¥thex¥n");
        }
        total = head.a_text + head.a_data
                    + head.a_bss;
        (void)printf("%lu¥t%lu¥t%lu¥t%lu¥t%lx",
                                  (u_long)head.a_text,
            (u_long)head.a_data, (u_long)head.a_bss,
                                        total, total);
        if (count > 1)
                (void)printf("¥t%s", name);
        (void)printf("¥n");
        return (0);
}
```

リスト 6　struct exec の定義（imgact_aout.h より抜粋）

```
/*
 * Header prepended to each a.out file.
 * only manipulate the a_midmag field via the
 * N_SETMAGIC/N_GET{MAGIC,MID,FLAG} macros in a.out.h
 */

struct exec {
     unsigned long a_midmag; /* flags<<26 | mid<<16 | magic */
     unsigned long a_text;   /* text segment size */
     unsigned long a_data;   /* initialized data size */
     unsigned long a_bss;    /* uninitialized data size */
     unsigned long a_syms;   /* symbol table size */
     unsigned long a_entry;  /* entry point */
     unsigned long a_trsize; /* text relocation size */
     unsigned long a_drsize; /* data relocation size */
};
/* XXX Hack to work with current kern_execve.c */
#define a_magic a_midmag
```

リスト 7
hello の先頭部分の 16 進ダンプ

```
% hexdump -C hello | head -n 3
00000000  7f 45 4c 46 01 01 01 09  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 03 00 01 00 00 00  80 83 04 08 34 00 00 00  |............4...|
00000020  7c 07 00 00 00 00 00 00  34 00 20 00 06 00 28 00  ||.......4. ...(.|
%
```

は，ヘッダに格納されている情報を読み，表示しているだけだということになります．

本当にそうなのでしょうか．実行形式をダンプして，実際に見てみましょう．リスト 7は，実行形式 hello の先頭部分の16進ダンプです．

ここでおかしなことに気がつきます．リスト 7を見て，先頭16バイトを struct exec に当てはめて解釈すると，テキスト領域のサイズは 0x09010101 バイト(リトル・エンディアンであることに注意)というとんでもなく大きな値になり，データ領域と BSS 領域のサイズは，0バイトになってしまっています．これはあきらかに変なので，実行形式の先頭 16バイト部分には，先に説明したような値は入っていないように見えます．さらに，リスト 7のテキスト表示の部分(右端)には，何やら“ ELF”という文字列が現れています．

では，この size のソースは何者なのでしょうか．ソースがどのように利用されるのかを知りたいときには，Makefile を見るのが一番です．/usr/src/usr.bin/size/Makefile を見てみると，

```
BINDIR= /usr/libexec/aout
```

となっています．これから，この size コマンドは，/usr/libexec/aout にインストールされるということがわかります．実はソース・コードをちゃんと読むとわかるのですが，/usr/src/usr.bin/size は，a.out 用の古い size コマンドのソース・コードなのです．struct exec は，a.out 形式のヘッダの構造体です．つまり，/usr/src/usr.bin/size にある size コマンドは，もともと a.out 形式の実行ファイルのテキスト，データ，BSS 領域のサイズを出力するためのものだったことがわかります．また，struct exec を見ればわかるように，a.out 形式では，セクションはテキスト，データ，BSS の三つしかないという前提で，決めうちになっています．これが，a.out 形式では，3種類の領域しかとることができない理由です．

FreeBSD の実行形式のフォーマットは，以前は a.out 形式でしたが，現在ではアプリケーション・プログラム，カーネルともに，デフォルトで ELF 形式です．このため，リスト 4で size コマンドを利用した際には，ELF 形式用の size コマンドが実行されたはずです．では，ELF 形式用の size コマンドはどこにあるのでしょうか．

FreeBSD では，size コマンドは現在では GNU binutils の一部として配布されているものを利用しているため，ソース・コードは /usr/src/gnu/usr.bin/binutils/size/usr/src/contrib/binutils/binutils/size.c にあります．また，その本体は，/usr/libexec/elf にあります．

リスト 8は，/usr/src/contrib/binutils/binutils/size.cからの抜粋です．ELF の実行形式の，それぞれのセクションのサイズをカウントしている部分です．

リスト 8のberkeley_sum()では，次のようにしてカウントしています．

1) ALLOCフラグが立っていないセクションは，無視する
2) ALLOCフラグが立っていて，CODEフラグかREADONLYフラグが立っているものは，テキスト領域としてカウントする
3) テキスト領域ではないが，CONTENTSフラグが立っているものは，データ領域としてカウントする
4) それ以外はBSS領域としてカウントする

リスト 8　/usr/src/contrib/binutils/binutils/size.cより抜粋

```
static bfd_size_type bsssize;
static bfd_size_type datasize;
static bfd_size_type textsize;

static void
berkeley_sum (abfd, sec, ignore)
     bfd *abfd ATTRIBUTE_UNUSED;
     sec_ptr sec;
     PTR ignore ATTRIBUTE_UNUSED;
{
  flagword flags;
  bfd_size_type size;

  flags = bfd_get_section_flags (abfd, sec);
  if ((flags & SEC_ALLOC) == 0)
    return;

  size = bfd_get_section_size_before_reloc (sec);
  if ((flags & SEC_CODE) != 0 || (flags & SEC_READONLY) != 0)
    textsize += size;
  else if ((flags & SEC_HAS_CONTENTS) != 0)
    datasize += size;
  else
    bsssize += size;
}
```

ここでもう一度，リスト 4(p.150)を見てください．20個のセクションには，それぞれにCONTENTS，ALLOC，LOADなどのフラグが設定されています．0～10番のセクションは，CODEフラグかREADONLYフラグが立っているため，テキスト領域としてカウントされます．11～16番のセクションは，CODEフラグもREADONLYフラグも立っていないが，CONTENTSフラグは立っているため，データ領域としてカウントされます．17番目のセクションは，ALLOCフラグだけのため，BSS領域となります．18，19番目のセクションは，ALLOCフラグが立っていないため，無視されます．これらのカウントの合計が，sizeコマンドの出力となります(リスト 4のsizeコマンドの出力では10進表示だが，リスト 5のsizeの項は16進表示になっていることに注意)．

つまり，ELF 用のsizeコマンドの場合には，領域のサイズは，複数のセクションの「集計値」になります．FreeBSD が扱うオブジェクト・フォーマットは，以前はa.out形式であったため，sizeコマンドはもともとはa.out形式を期待して，テキスト，データ，BSSの三つの領域を表示していました．しかしその後，オブジェクト・フォーマットはELF形式に変更されました．ELF形式では任意の数のセクションを持たせることができるため，フラグ情報を見てテキスト，データ，BSSのいずれかに分類するようなしくみになっていると思われます．

なお，ELF形式では，複数の「セクション」をまとめたものを「セグメント」として定義することができます[2]．セグメントは，ELF形式のファイル中に「プログラム・ヘッダ」として記述

リスト 9
objdump -p helloの結果

```
% objdump -p hello

hello:     file format elf32-i386

Program Header:
    PHDR off    0x00000034 vaddr 0x08048034 paddr 0x08048034 align 2**2
         filesz 0x000000c0 memsz 0x000000c0 flags r-x
  INTERP off    0x000000f4 vaddr 0x080480f4 paddr 0x080480f4 align 2**0
         filesz 0x00000019 memsz 0x00000019 flags r--
    LOAD off    0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12
         filesz 0x0000050d memsz 0x0000050d flags r-x
    LOAD off    0x00000510 vaddr 0x08049510 paddr 0x08049510 align 2**12
         filesz 0x000000d0 memsz 0x000000ec flags rw-
 DYNAMIC off    0x00000520 vaddr 0x08049520 paddr 0x08049520 align 2**2
         filesz 0x00000098 memsz 0x00000098 flags rw-
    NOTE off    0x00000110 vaddr 0x08048110 paddr 0x08048110 align 2**2
         filesz 0x00000018 memsz 0x00000018 flags r--

Dynamic Section:
  NEEDED      libc.so.4
  INIT        0x8048334
  FINI        0x80484f8
  HASH        0x8048128
  STRTAB      0x804827c
  SYMTAB      0x804817c
  STRSZ       0x9d
  SYMENT      0x10
  DEBUG       0x0
  PLTGOT      0x80495c8
  PLTRELSZ    0x18
  PLTREL      0x11
  JMPREL      0x804831c

%
```

リスト 10　さまざまな変数の定義(values_sub.c)

```
char c = 'a';
int i0;
int i1 = 0;
int i2 = 1;
static int si0;
static int si1 = 0;
static int si2 = 1;
const int ci0;
const int ci1 = 0;
const int ci2 = 1;
char * p0;
char * p1 = "sample0";
char str[] = "sample1";

int func()
{
  int x = 0;
  return (x);
}
```

リスト 12　nm values_sub.o の結果

```
% nm values_sub.o
00000000 D c
00000004 C ci0
00000000 R ci1
00000004 R ci2
00000000 T func
00000000 t gcc2_compiled.
00000004 C i0
00000004 D i1
00000008 D i2
00000004 C p0
00000014 D p1
00000000 b si0
0000000c d si1
00000010 d si2
00000018 D str
%
```

リスト 11　さまざまな変数の定義(values.c)

```
#include <stdio.h>

extern char c;
extern int i0, i1, i2;
extern const int ci0, ci1, ci2;
extern char *p0, *p1;
extern char str[];

int main()
{
  printf("   c = %c¥n", c);
  printf("  i0 = %d,   i1 = %d,   i2 = %d¥n",  i0,  i1,  i2);
  printf(" ci0 = %d,  ci1 = %d,  ci2 = %d¥n", ci0, ci1, ci2);
  printf("  p1 = %s,  str = %s¥n", p1, str);
  printf("  &c = 0x%08x¥n", &c);
  printf(" &i0 = 0x%08x,  &i1 = 0x%08x,  &i2 = 0x%08x¥n",
                                              &i0,  &i1,  &i2);
  printf("&ci0 = 0x%08x, &ci1 = 0x%08x, &ci2 = 0x%08x¥n",
                                              &ci0, &ci1, &ci2);
  printf(" &p1 = 0x%08x, &str = 0x%08x¥n", &p1, &str);
  exit (0);
}
```

されています．実行形式のロード時には，プログラム・ヘッダを見て，セグメント単位で展開することで，似たような属性のセクションをまとめて展開することができます．

プログラム・ヘッダの情報は，objdump -pで知ることができます(リスト 9)．またobjdumpは，--all-headersですべてのヘッダ情報を表示します．

なおobjdumpコマンドは，実行形式，オブジェクト・ファイル，ダイナミック・リンク・ライブラリ，コア・ダンプのライブラリ・アーカイブに対して実行可能です．

シンボルと再配置

● シンボルとはなにか

C言語のソース上で変数を定義すると，その本体が必ずどこかに作成されます．メモリ的な観点からいうと，「利用可能なメモリ空間上のどこかのアドレスに，変数用の領域が確保される」ということになります．C言語上での変数へのアクセスは，実際には当該のメモリへのアクセスになります．このため，変数名は当該のメモリ領域のエイリアス(別名)であるということもできるでしょう[注5]．これらは，関数についても同様のことが言えます．

ここで，values_sub.c(リスト 10)，values.c(リスト 11)のようなプログラムを考えてみます．values_sub.cでは，さまざまな変数を定義しています．values.cでは，それらの変数を参照しています[注6]．

試しに，values_sub.cをコンパイルしてみます．gccは-cオプションをつけて実行することで，リンクを行わず，コンパイルだけを行い，オブジェクト・ファイル(ここでは，values_sub.o)を作成します．なお，コンパイルは最適化なし(-Oオプションなしのデフォルト動作)で行っています．

```
% gcc -c values_sub.c
% ls values_sub.o
values_sub.o
%
```

value_sub.cがコンパイルされ，オブジェクト・ファイルvalues_sub.oが作成されました．オブジェクト・ファイルに存在している関数や変数の情報は，nmコマンドで確認することができます．リスト 12は，values_sub.oに対してnmを実行したときの出力結果です．

リスト 12では，関数func()とそのほかさまざまな変数の情報が表示されています．関数も変数も，オブジェクト・ファイル上では，単なる「名前」として扱われます．この「名前」のことを「シンボル」と呼びます．リスト 13のようなシンボルの一覧は，「シンボル・リスト」，「ネーム・リスト」などと呼ばれます．リスト 12では，各種のシンボルに対して，DやCなどのタイプが指定されていることがわかります．

シンボルを管理するための「シンボル情報」は，関数や変数の定義ごとに存在し，シンボル名，シンボルのタイプ，シンボルの実体の位置などの情報を保持しています．オブジェクト・ファイルは，シンボルを管理するために，シンボル情報の配列として「シンボル・テーブル」を持っています．nmコマンドは，ファイル中のシンボル・テーブルを検索し，その一覧を出力するためのコマンドです．

なお，シンボル情報の配列がシンボル・テーブルと呼ばれる

注5：変数がレジスタに直接割り当てられている場合には話は違ってくる．
注6：本来はヘッダ・ファイルを作成するべきだが，本稿では省略してある．ヘッダ・ファイルの作成に関しては，参考文献 1)を参照．

リスト 13　nm values の結果

```
% nm values
080497a0 A _DYNAMIC
08049848 A _GLOBAL_OFFSET_TABLE_
0804983c ? __CTOR_END__

        ...(中略)...

08049770 D __progname
         w __register_frame_info
08049860 A _edata
0804988c A _end
080485d8 T _fini
08048334 T _init
08048380 T _start
         U atexit
0804977c D c
08049884 B ci0
08048760 R ci1
08048764 R ci2
08049778 d completed.4
0804988c A end
0804987c B environ
         U exit
0804846c t fini_dummy
0804979c d force_to_data
0804977c d force_to_data
08048474 t frame_dummy
08048590 T func
08048590 t gcc2_compiled.
08048418 t gcc2_compiled.
080484a0 t gcc2_compiled.
080485a8 t gcc2_compiled.
08049888 B i0
08049780 D i1
08049784 D i2
08048498 t init_dummy
080485d0 t init_dummy
080484a0 T main
08049860 b object.11
08049774 d p.3
08049880 B p0
08049790 D p1
         U printf
08049878 b si0
08049788 d si1
0804978c d si2
08049794 D str
%
```

ことに対して，単一のシンボル情報のことを指すことばはとくにないようですので，本稿では「シンボル・テーブル・エントリ」と呼ぶこととします．

● シンボルの実例

values.cにvalues_sub.oをリンクして，実行形式valuesを作成してみましょう．gccは先にも説明したとおり，実行形式作成用の「コンパイラ・ドライバ」なので，引き数にオブジェクト・ファイルやソース・コードを複数指定しても，適切に扱ってコンパイルとリンクを行ってくれます．

```
% gcc values.c values_sub.o -o values
%
```

ここではソース・ファイルであるvalues.cと，オブジェクト・ファイルであるvalues_sub.oを同時にgccに与えていますが，実際にはvalues.cは自動的にコンパイルされてオブジェクト・ファイルが作成されます．リンカにはオブジェクト・ファイルが渡され，リンクが行われて実行形式が作成されます．オブジェクト・ファイル中には，実行コードや変数値のデータなどが格納されていますが，それらが実際にメモリ上の固定アドレスに割り当てられるのは，リンク時です．このアドレス割り当て作業のことを，再配置（relocation）と呼びます．再配置によって，関数や変数の実体は，特定のアドレスに配置されます．再配置はすべてのシンボルが出そろった状態でないと行えないため，すべてのオブジェクト・ファイルとライブラリが出そろった，最後のリンクの段階で行われる必要があります．このためリンクのことをあえて「最終リンク」と呼ぶこともあります．

再配置のためには，関数や変数の実体が，どのようなアドレスに配置されてもかまわないような構造になっている必要があります．これを再配置可能（relocatable）と呼びます．このためオブジェクト・ファイルのことを，再配置可能ファイル（relocatable file）と呼ぶこともあります．実行コードが再配置可能であるということは，当該のコードを任意のアドレスに配置できるということです．

再配置可能であるためには，次の二つの条件を満たしている必要があります．まず一つ目として，関数や変数は，実際にどこに配置されるのかが不定なので，シンボル名とその実体のファイル中での位置の対応表が必要になります．このためにシンボル・テーブルが必須になります．

二つ目として，実行コード中には，いたるところに関数呼び出しやグローバル変数の参照が存在します．これらは実際には，当該アドレスへのジャンプや参照になります．コンパイル時にはこれらのアドレスは未定なので，再配置可能であるためには，これらは「未解決シンボル」として，実際のアドレスの部分が空欄となっており，後で補填できるように，補填するための情報を別に保持している必要があります．これを「再配置情報」と呼びます．シンボル情報が関数や変数の「定義」，「実体」ごとに存在することに対して，再配置情報は，関数や変数の「呼び出し」，「利用」ごとに存在します．つまりこれらは「1対n」の対応関係になります．

再配置情報の一つ一つは，「再配置エントリ」と呼ばれます．再配置可能ファイルは，再配置エントリの配列をもっている必要があります．なお再配置エントリに対して，再配置エントリの配列を指すことばはとくにないようなので，本稿では「再配置テーブル」と呼ぶこととします．

● リンカの動作

リンカはまず，関数や変数の実体が存在するセクションを，実際のメモリのアドレス上に割り当てます．これにより，関数や変数の実体には，アドレスが割り当てられることになります．次に，シンボル・テーブルに登録されている関数や変数が，どのアドレスに配置されたかのデータベースを作成します．これはシンボル・テーブルに対して行われます．さらに，再配置テーブルを参照して，関数を呼び出したり変数を利用している部分に，実際のアドレスを補填します．この時点で関数や変数の実体と利用側は，シンボルを経由せず，直接に結合されます．これを「名前解決」と呼びます．

リンクして作成された実行形式valuesに対して，nmを実行してみましょう（リスト13）．リンク後にもシンボル・テーブルは残されるため，nmはリンク後の実行形式に対しても行う

表1　シンボルのタイプ一覧（主要なもの）

タイプ	内　容
B	未初期化のためBSS領域に割り当てられる
D	初期化済みのためデータ領域に割り当てられる
R	読み込み専用の領域に割り当てられる

ことができます．これは，デバッガを利用したデバッグ時に，関数名や変数名などを利用できるようにするためです．

リスト13では，変数が割り当てられているアドレスと，タイプが表示されています．たとえば変数cのアドレスは0x0804977cに割り付けられており，タイプはDです．また，変数i0は，0x08049888に割り付けられており，タイプはBです．

これらのタイプの意味は，info nmを読むとわかります[注7]．表1は，主要なタイプの意味をinfo nmから抜粋してまとめたものです．なお，リスト14で小文字のものはローカル（別のファイルからはリンクできない），大文字のものはグローバル（別のファイルからもリンク可能）なシンボルになります．

リスト13では，変数c，i1，i2はタイプがDとなっています．values_sub.cでは，これらは初期値をもつ変数として定義してあるので，データ領域に割り付けられることになります．これらのアドレスが実際にどのセクションに割り付けられているのかは，objdumpでの結果と見比べることで確認できます．

リスト14は，実行形式valuesに対してobjdump -hを実行したときの結果です．たとえば変数c，i1，i2は，リスト13より，0x0804977c～0x08049784の範囲に割り当てられていることがわかります．これはリスト14では.dataセクションに相当するので，データ領域に割り当てられていることがわかります．.dataセクションはREADONLYフラグは立っていないため，これらの変数には書き込みが可能です．

リスト15は，valuesの実行結果です．リスト13で表示されているc，i1，i2のアドレスと，リスト15で表示されているc，i1，i2のアドレスが，一致していることがわかります．

リスト13では，ci1，ci2のタイプはRとなっています．アドレスはそれぞれ0x08048760，0x08048764となっています．リスト14によれば，これらは.rodataセクションに配置されています．.rodataはその名のとおりread onlyなデータが割り当てられるセクションです．values.cではci1，ci2はconstな変数として定義してあるので，書き込みのできないセクションに割り当てられているわけです．

リスト13では，ci0，i0，p0のタイプはBになっています．これらのアドレスは，0x08049880～0x08049888になります．リスト14では，これらは.bss領域に割り当てられています．values.cではci0，i0，p0は初期値なしの変数として定義されているため，BSS領域に配置されているわけです．

リスト14　objdump -h valuesの結果

```
% objdump -h values

values:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn

        ...(中略)...

  8 .text         00000258  08048380  08048380  00000380  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  9 .fini         00000006  080485d8  080485d8  000005d8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 10 .rodata       00000190  080485e0  080485e0  000005e0  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 .data         0000002c  08049770  08049770  00000770  2**2
                  CONTENTS, ALLOC, LOAD, DATA

        ...(中略)...

 17 .bss          0000002c  08049860  08049860  00000860  2**2
                  ALLOC

        ...(後略)...
```

リスト15　valuesの実行結果

```
% ./values
   c = a
  i0 = 0,   i1 = 0,   i2 = 1
 ci0 = 0,  ci1 = 0,  ci2 = 1
  p1 = sample0,  str = sample1
  &c = 0x0804977c
 &i0 = 0x08049888,  &i1 = 0x08049780,  &i2 = 0x08049784
&ci0 = 0x08049884, &ci1 = 0x08048760, &ci2 = 0x08048764
 &p1 = 0x08049790, &str = 0x08049794
%
```

最後にmain()，func()です．これらは関数ですが，関数もやはりメモリ上に存在するため，アドレスを持っています．リスト13では，これらのタイプはTとなっており，アドレスは0x08049880～0x08049888になっています．リスト14によれば，これらは.text領域に割り当てられています．.textセクションは，テキスト領域に相当するセクションです．main()，func()は関数であるため，実行コードを含んでいます．このためテキスト領域に配置されているわけです．

シンボル・テーブルは，実行形式のオブジェクト中にも残されています．しかしシンボル・テーブルは，通常の実行時にはデバッガを利用しないならば不要です．このため，削除してしまってもかまいません．stripでシンボル・テーブルを削除することで，実行形式のファイル・サイズを節約することができます．

リスト16に示すように，実行形式valuesに対してstripを行うことで，ファイル・サイズが約65％程度に節約されていることがわかります．また，fileの結果は"not stripped"

注7：gccやbinutilsなどのGNUのツールは，manよりもinfoのほうが情報が充実している場合が多いので，困ったときにはmanだけでなく，infoも見てみるとよいだろう．

から“stripped”となり，nmによるシンボル・テーブルの表示は“no symbols”となって不可能になっています．

ローダの仕事

● ローダとはなにか

OSがプログラムを実行する際には，OSは実行形式について，以下のことを知る必要があります．

- どのようなセクション（セグメント）があるのか
- それぞれのセクション（セグメント）のサイズは
- どのアドレスに展開すべきか
- どのアドレスから実行すべきか

これらの情報は，すべて実行形式のヘッダ上に格納されています．プログラムを実行する際には，これらの情報を読み込み，与えられた情報に沿ってセクションをメモリ上に展開し，指定されたアドレスから実行を開始するような動作が必要になります．このような処理を行うプログラムを「ローダ」と呼びます．つまりローダの役割は，ELF形式などのフォーマットを理解して，実行形式が期待しているとおりに，メモリ上に展開することです．

通常のアプリケーション・プログラムの場合には，プログラムのロードと実行を行うのは，OSの役目です．実際にはアプリケーション側からexec()ファミリのシステム・コールを実行したときに，OSによってロードが行われます．

● カーネルのブートにもローダは必要

また，アプリケーションに限らず，OSのカーネルの場合にも，ローダは必要です．カーネルの本体は，コンピュータの電源を入れた瞬間からはじめからメモリ上に存在するわけではない[注8]ので，OSの機械語コードをRAM上に展開して実行を開始するようなしくみが必要になります．このためのローダのことを，「ブート・ローダ」，「カーネル・ローダ」などと呼びます．

FreeBSDのカーネルは，ELF形式でルート・ファイル・システム上に/kernelとして置かれています．カーネル・ローダは，これを読み込み，ELF形式を解釈して，メモリ上に展開します．よってFreeBSDのカーネル・ローダは，ファイル・システムとELF形式を解釈できる必要があります．これはそれなりのプログラム・サイズになってしまいますが，PCではBIOSの制約のため，最初から巨大なプログラムを実行することができません．そこでAT互換機では，低級なローダが少しずつ高級なローダを起動していくように，段階的にブートするようになっています．

FreeBSDやNetBSDのカーネルをビルドした際に作成されるのは，「カーネルそのもののELF形式」です．そして，このELFファイルが，ルート・ファイル・システムのトップに置かれています．したがって，ブート・ローダがELFファイルを物理メモリ上に展開して実行を渡すと，カーネルのトップからスタートすることになります．

しかしLinuxカーネルの場合には，そうではありません．Linuxカーネルについて，少し説明しましょう．

Linuxのカーネルをビルドする際には，make menuconfig，make dep，make clean，make zImage（もしくはbzImage），make installを順に行うのが，常套（じょうとう）の流れです[注9]．make (b)zImageを行うと，自動的にmake vmlinuxが行われ，vmlinuxというファイルが作成されます．これはカーネル本体のELF形式であり，上で説明したFreeBSDやNetBSDのカーネルのELFファイルに相当するものです．つまり，FreeBSD，NetBSDならば，この段階でカーネルのビルドは終了です．

しかしLinuxカーネルのmake (b)zImageでは，vmlinuxの作成後，もう一つELF形式が作成されます．たとえば，Linux/PowerPCでは，make zImageによりzImage.elfという ELF形式が作成されます．Linux/i386ならば，make bzImageによりbzImageという ELF形式が作成されます．ここではLinux/PowerPCのzImage.elfを例として説明しますが，make installを行った際に，実際にインストールされるのは，実はこのzImage.elfです．では，zImage.elfとは何者なのでしょうか．

結論から言うと，このzImage.elfは，カーネルのベタ・バイナリを特定セクションにイメージ・データとしてもっている，「カーネル・ローダ」です（このローダの作成方法に関しては後述）．つまり，Linux/PowerPCでは，カーネルは以下の手順で起動します（図3）．

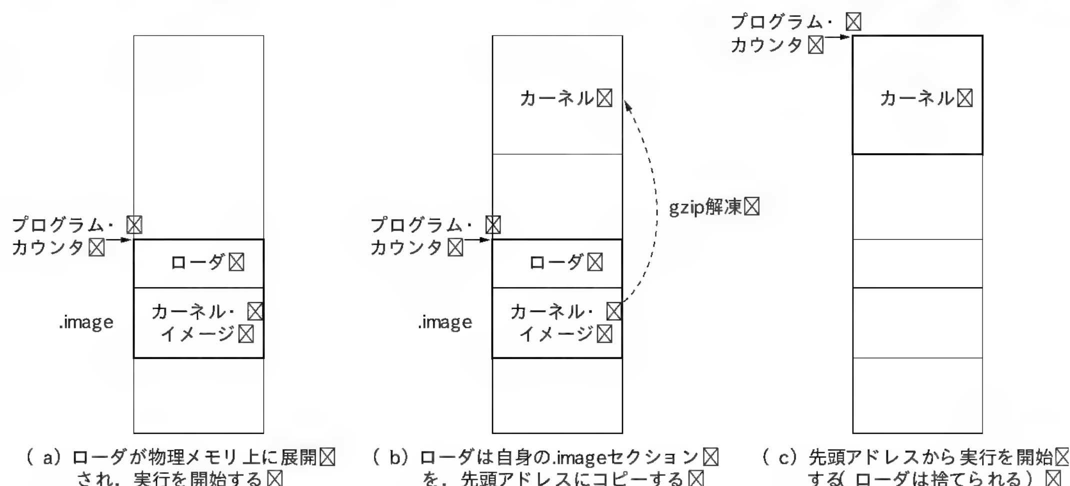
1) ローダがzImage.elf（カーネル・ローダ）を，物理メモリ上に展開する．実際のカーネルを展開するときに，カーネル・ローダ自身とぶつからないように，ある程度後のほうの物理メモリに展開する〔図3（a）〕
2) カーネル・ローダが動作を開始する
3) カーネル・ローダは自身の.imageセクションに格納されているカーネルのベタ・バイナリ（実際にはgzip圧縮されてい

リスト16　strip valuesの実行結果

```
% ls -l values
-rwxr-xr-x  1 hiroaki  user  5327  4  8 19:08 values
% file values
values: ELF 32-bit LSB executable, Intel 80386,
    version 1 (FreeBSD), for FreeBSD 4.8,
    dynamically linked (uses shared libs), not stripped
% strip values
% ls -l values
-rwxr-xr-x  1 hiroaki  user  3480  4  8 19:08 values
% file values
values: ELF 32-bit LSB executable, Intel 80386,
    version 1 (FreeBSD), for FreeBSD 4.8,
    dynamically linked (uses shared libs), stripped
% nm values
/usr/libexec/elf/nm: values: no symbols
```

注8：例外として，ROM上にOSを配置してあって，ROMのまま動作する，という場合がある（XIP：eXecute In Place）．

注9：2.6系のカーネルから，make depはなくなった．また，ここではモジュールの作成とインストールは省略してある．



（a）ローダが物理メモリ上に展開され，実行を開始する　（b）ローダは自身の.imageセクションを，先頭アドレスにコピーする　（c）先頭アドレスから実行を開始する（ローダは捨てられる）

図3　Linux/PowerPCのブート手順

る）を，物理メモリ上に展開する〔図3（b）〕
4）展開したカーネルに処理を移す〔図3（c）〕
5）カーネルが動作を始める

つまり，zImage.elfの実体は，巨大なバイナリ・データをもつ，ローダなのです．Linuxがこのような2段階ブートになっているのは，カーネルを圧縮することで，zImage.elfのサイズを小さくするためのようです．

ここで言っている「ベタ・バイナリ」というのは，カーネルを物理メモリ上に展開した際の，そのままの「ベタな」イメージのことです．またLinux/PowerPCでは，ベタ・バイナリは，物理メモリの先頭アドレス（0番地）にロードされます．PowerPCを含む通常のCPUだと，0番地には割り込みベクタ（もしくは割り込み処理ルーチン）が存在する場合が多々ありますが，これでは割り込みベクタの上にカーネルがロードされてしまうことになります．なぜ，それでも問題ないかというと，Linux/PowerPCでは，実はカーネルのビルドの際に，これらの割り込みベクタもはじめから展開された状態（アセンブラでそのように書かれている）でカーネルを作成し，ベタ・バイナリにしているのです．したがって，ベタ・バイナリのロードと同時に割り込みベクタも展開され，カーネルが起動する際には，すでに割り込みベクタも用意されていることになります．このため，起動時の割り込みベクタのインストールなどは行われないようです[注10]．

なお，FreeBSDやNetBSDでは，カーネルの起動時に，カーネル自身が割り込みベクタをインストールするので，このような問題はありません．

注10：筆者はLinuxについてはあまりよく知らないのだが，少なくともLinux/PowerPCでは，カーネルのコードは0番地に置かれていることが前提で書かれており，移動するのはめんどうそうだ．筆者はLinuxのソースを読んでこれらの事実を知ったとき，いろいろな意味で衝撃的だった．

リスト17　リンカ・スクリプトの雛型

```
% ls /usr/libdata/ldscripts
elf_i386.x      elf_i386.xc     elf_i386.xr     elf_i386.xsc
elf_i386.xbn    elf_i386.xn     elf_i386.xs     elf_i386.xu
%
```

リンカ・スクリプト

ELF形式では，任意数のセクションをもてたり，セクションにフラグをつけることができます．また，VMAやLMAを指定することができます．

実行形式を作成する際には，これらのさまざまなチューニングが行われることが考えられます．しかし，これらのパラメータをすべてコマンド・ライン・オプションで指定することは現実的ではありませんし，それでは融通が効きません．このため，通常はリンカは，これらのパラメータを設定ファイルに記述し，設定ファイルを読み込んで，それに応じてリンクを行うような動作を行います．この設定ファイルのことを「リンカ・スクリプト」と呼びます．

FreeBSDでは，/usr/libdata/ldscripts以下にリンカ・スクリプトのひな型があります（リスト17）．ここには，通常の実行形式用や，共有ライブラリ用のリンカ・スクリプトがあります．リンカ・スクリプト中のコメントを見た限りでは，elf_i386.xが通常の実行形式用のリンカ・スクリプトのようです．

また，OSのカーネルのリンク時には，通常のアプリケーションとは違ったリンク方法が必要になることが多々あります．たとえば，仮想記憶を持つOSの場合には，VMA ≠ LMAとなることが多く，このような場合には，専用のリンカ・スクリプトが必要になります（後述するobjcopyで調整することもできる）．FreeBSDのカーネル（i386版）の場合には，/usr/src/

リスト 18　エントリ・ポイントについて(info ldより抜粋)

```
Setting the entry point
-----------------------

   The first instruction to execute in a program is called
 the "entry point".  You can use the `ENTRY' linker script
 command to set the entry point.  The argument is a symbol
 name:
     ENTRY(SYMBOL)
```

リスト 19　ENTRY( )の指定(elf_i386.xより抜粋)

```
/* Default linker script, for normal executables */
OUTPUT_FORMAT("elf32-i386", "elf32-i386",
              "elf32-i386")
OUTPUT_ARCH(i386)
ENTRY(_start)
SEARCH_DIR("/usr/lib");
... 後略 ...
```

sys/conf/ldscript.i386が利用されます(/usr/src/sys/conf/Makefile.i386参照).

残念ながら，リンカ・スクリプトに関する和文の資料はあまりありません．本稿ではリンカ・スクリプトの文法については説明しませんが，info ldに詳しい説明があるので，興味のある方は，そちらを参照してください．リスト 18のファイルの内容も，参考になるかもしれません．

スタートアップ・ルーチン

リンカ・スクリプトは各セクションの配置情報などを指定しますが，プログラムの開始アドレスも，リンカ・スクリプトによって指定することができます[注11]．info ldの，Scripts - Simple Commands - Entry Pointという章には，リスト 18のような説明があります．リスト 18によれば，プログラムの実行開始位置は「エントリ・ポイント」と呼ばれ，エントリ・ポイントは，リンカ・スクリプト中でENTRY()というコマンドで指定できるとあります．

リスト 19は，/usr/libdata/ldscripts/elf_i386.x(実行形式用のデフォルトのリンカ・スクリプト)の先頭部分の抜粋です．5行目にENTRY(_start)という行がありますが，これが，プログラムの起動時に実行が開始される位置です．したがって，プログラムの実行時に，本当にいちばん最初に実行されるのは，main()ではなく，_startです．つまり，_start()という関数(正確にはシンボル)が必要になってきます．しかし，我々がC言語によってプログラムを書く際には，_start()のような関数を明示的に作成することはありません．では，_start()はどこにあるのでしょうか．

最初のほうに出てきたリスト 2では，最終リンクの段階で，crt1.o，crti.o，crtbegin.o，crtend.o，crtn.oという五つのオブジェクト・ファイルが，暗黙のうちにリンクされています．これらのソース・コードは，/usr/src/lib/csu以下にあります．実はこれらのソースを見るとわかるのですが，_startというシンボルは，crt1.cの中にあります．csuは“C Start Up”のことです．

_startでは，main()への引き数の用意や，レジスタの初期化，スタックの設定などが行われます．プログラムを実行したときのOSの役割は，そのプログラムをメモリ上に展開し，制御を渡すことだけです．このため，レジスタの初期化などは，プログラム側の責任になります．しかし，これらは定型の作業になりますし，アセンブラの知識なども必要になってくるため，スタートアップ・ルーチンとしてまとめられ，自動的にリンクされるようになっています．つまり一般のC言語ユーザは，これらのことはまったく気にすることはなく，「プログラムはargc，argvを引き数として，main()から始まる」と思っていればいいことになっているのです．

まとめ

アプリケーションを書くうえでは，普段あまり意識していないリンカとローダですが，今回は第1回であることから基礎的な知識と用語をおもに説明しました．

次回は，代表的なオブジェクト・フォーマットであるELF形式と，ライブラリ・アーカイブについて，内部構造などを具体的に説明します．

参考文献

(1) 坂井 弘亮;「C言語 入門書の次に読む本」，技術評論社
(2) John R. Levine 著，榊原一矢監訳;「Linkers & Loaders」，ポジティブエッジ訳
(3) Peter van der Linden 著，梅原 系訳;「エキスパートCプログラミング――知られざるCの深層」，アスキー
(4) 西田 亙;「ゼロから始める組み込みLinuxシステム構築」，Embedded UNIX Vol.6.

注11：GNU ldの-eオプションで指定することもできる．

さかい・ひろあき　富士通アイ・ネットワークシステムズ

組み込み向けストレージ・デバイスとして注目される

フラッシュ・メモリの基礎と電源障害に強いファイル・システムの構築

長澤 恒也

2000年頃から，NAND型フラッシュ・メモリ(以下，フラッシュ)をオンボードに搭載して，データ・ストレージとして使用するシステムが増えてきました．NAND型フラッシュは，その構造上の特性からハードディスクの置き換えを狙って製品開発が進められています．

NAND型フラッシュは日本人が発明した世界に誇れるメモリ素子です．携帯電話やディジタル家電などのマーケットがどんどん大きくなっていくなかで，半導体の製造プロセスが進歩し，集積度が高いNAND型フラッシュはデータ・ストレージ・デバイスの要として，ますます重要な位置を占めるようになってきました．

本稿では，このNAND型フラッシュのデバイスの特徴を踏まえながら，データ・ストレージとして利用する際の要点をまとめて解説すると同時に，フラッシュの特性を活かした電源障害に強いファイル・システムの内部構造にも触れてみます．

フラッシュとは

フラッシュの最大の特徴は不揮発性であることです．つまり，いったん記憶させたデータは，電源が切れた後も消えてなくならず，再度電源を入れればそのまま読み出すことができます．このような特徴を持ったメモリ全般を不揮発性メモリと呼びます．

フラッシュ以外にも不揮発性メモリにはいろいろな種類があり，それぞれのデバイスには，書き込み/消去の方法に特徴があります(表1)．

もっとも単純な不揮発性メモリは，マスクROMです．これは半導体の製造時点で記憶するデータを専用のマスク・パターンとして作ることからそう呼ばれています．半導体工場で大量に製造できるので製造コストが安くなり，量産数量の多い民生機器などにはもってこいなのですが，一度作ってしまうとデータを変更するためにたいへんなコストと時間がかかってしまいます．したがって，用途は非常に限られてしまいます．

紫外線消去型のEPROM(Erasable Programmable ROM)はセラミック・パッケージの真ん中に石英ガラスの窓が付いた形をしています．データの書き込みは専用のプログラマ(書き込み専用装置)を使って電気的に行い，書き込まれたデータを消去するのに紫外線を用います．パッケージ中央のガラスの奥に見えているメモリ・チップに紫外線を照射すると，メモリ・セルの記憶が消去されます．EPROMの書き込み/消去はメモリをプリント基板から取り外して行うので，それなりの手間がかかり，大量生産時のスループットを落とす要因にもなります．

EEPROM(Electrically Erasable PROM)は電気的に消去/書き込み可能な不揮発性メモリです．紫外線の助けを借りることなく，高電圧によって消去が可能になっています．EEPROMのメモリ・セルは，選択用のトランジスタとメモリ用のトランジスタが組み合わされており，各ビットごとに書き込み/消去がで

表1 さまざま不揮発性メモリ

不揮発性メモリ	消 去	書き込み	読み出し
マスクROM	不可	マスク・パターン 書き換え不可	ランダム・アクセス
UV-EPROM	紫外線照射 チップ一括消去	ランダム・アクセス ビット単位	ランダム・アクセス ビット単位
EEPROM	ビット単位	ランダム・アクセス ビット単位	ランダム・アクセス ビット単位
NOR型フラッシュ	ブロック単位 1s	ランダム・アクセス ビット単位 10～100μs/バイト	ランダム・アクセス ビット単位 数10ns/バイト
NAND型フラッシュ	ブロック単位 2～4ms	シーケンシャル・アクセス ページ単位 200μs/ページ＋転送時間	シーケンシャル・アクセス ページ単位 15～25μs/ページ＋転送時間

きます．しかし，1ビットの記憶に二つのトランジスタを用いるために，DRAMのように集積度を上げることができません．

NOR型のフラッシュはメモリ・セルごとに消去をしないかわりに，選択用トランジスタを不要にし，1ビットあたり1トランジスタでメモリ・セルが構成されています(図1)．そのため消去は一括消去となりますが，1セルあたり1トランジスタ＋1キャパシタのDRAMよりも集積度を上げることができるようになりました．

NAND型フラッシュは，さらにNOR型フラッシュのような1ビットごとのアドレッシングをせず，特定のビット線から順番にデータが読み出せるような構造にすることで配線面積を抑え，最小のメモリ・セルを実現したメモリです．

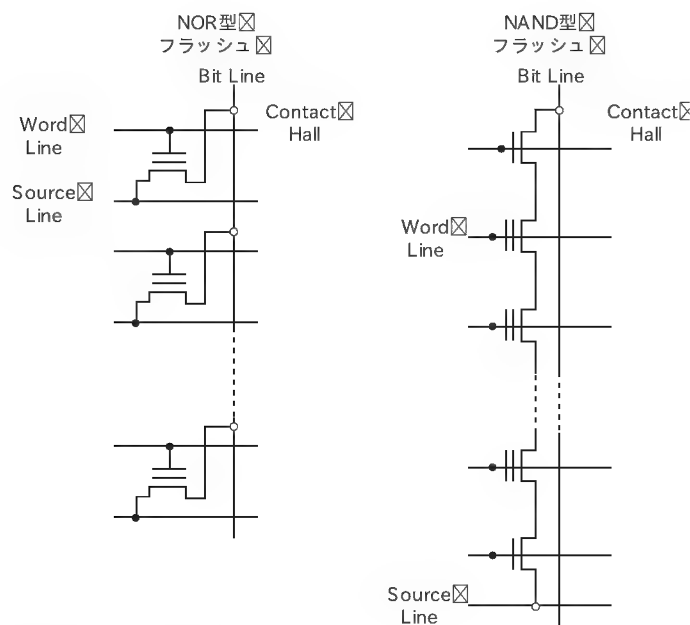


図1 メモリ・セルの模式図



図2 ブロックとページの構成

NAND型フラッシュの読み出し/書き込み操作

● フラッシュのプログラム・モデル

NAND型フラッシュの大きな特徴として，読み出しや書き込みを，シーケンシャルに実行する点があげられます．メモリにアクセスするには，まず所定のコマンドを投入してから，メモリのアドレスを必要サイクル分投入します．そのうえで必要なデータの読み出しや書き込みが実行できます．NOR型フラッシュのようにメモリ空間上にメモリ・セルがマッピングされ，ランダム・アクセスできる構造にはなっていません．この点はメモリというより，むしろHDDなどに近いアクセス方式となっています．

フラッシュの読み書きはページと呼ばれる528(512＋16)バイト単位で実行します．1ページは，512バイトのデータ部と16バイトの冗長部に分けられます．データ部には通常のデータを格納し，冗長部にはECCデータや不良ブロック・マーク，そのほかの管理情報を格納するのが一般的な使い方です．ちなみにデータ部と冗長部のメモリ・セルには何ら違いはなく，このように使い分けると便利なように内部の制御回路が構成されており，それにあわせてコマンドが用意されているにすぎません．データ部よりも冗長部のほうが不良率が低いといった差はありません．

ブロックはフラッシュの一括消去の単位で，32ページ(16Kバイト)で構成されています．ページとブロックにはそれぞれアドレスとして0から番号が割り当てられています(図2)．

● 制御信号

フラッシュへの読み書きはすべて8ビットないしは16ビットのデータ・ポートを経由して行われます．データ・ポートで読み書きされるデータを区別するために，入出力用の制御信号として_CLE(Command Latch Enable)，_ALE(Address Latch Enable)，_RE(Read Enable)，_WE(Write Enable)があり，この組み合わせでコマンド，アドレス，データ(読み込み/書き込み)を区別します．NAND型フラッシュの制御信号一覧を表2に示します．

多くのデバイスと同様に_CE(Chip Enable)があり，バス・マスタのフラッシュ・チップへのアクセスを明示します．フラッシュの内部状態を示すRY/_BY信号もあり，ドライバ・ソフトウェアでこの信号レベルを監視してチップの動作状態を判別します．

NAND型フラッシュは上記のとおり，CPUから見た場合にはメモリというよりもパラレル・ポート付きのI/Oデバイスのように見えます．フラッシュをCPUに接続する場合には，何通りかの方法があります．

1) すべての制御信号，データ・ポートをGPIOに接続

すべての制御線とデータ・ポートをGPIOに接続し，チップ

表2　NAND型フラッシュの制御信号一覧

_CE	チップ・イネーブル
_WE	ライト・イネーブル
_RE	リード・イネーブル
CLE	コマンド・ラッチ・イネーブル
ALE	アドレス・ラッチ・イネーブル
_WP	ライト・プロテクト
RY/_BY	レディ・ビジー出力
I/O	コマンド・アドレス・データ入出力

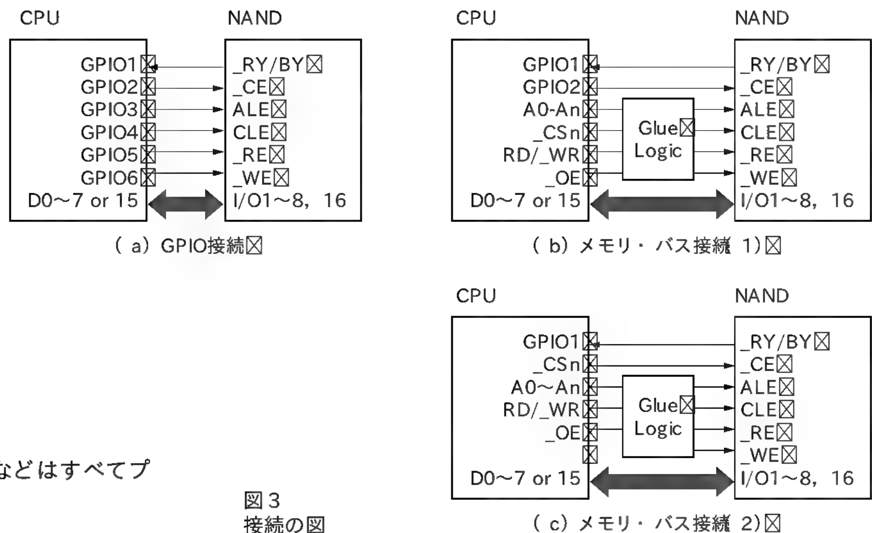


図3
接続の図

の選択や読み込み/書き込みのタイミング制御などはすべてプログラムで実現します〔図3(a)〕．

2) メモリ・バスに接続

CPUのメモリへのリード/ライトでそのままデータ・ポートを読み書きできるように，簡単なグルー・ロジックを介して接続する方法です．メモリ・バスの制御信号で_ALE，_CLE，_RE，_WEを作り，_CEとRD/_BYのみをGPIOに接続します〔図3(b)〕．

NAND型フラッシュには"CE Don't Care"というタイプが存在します〔図3(c)〕．通常のNANDフラッシュでは，コマンド投入からデータの読み書き完了まで，_CEピンをアサートし続けていなければいけません．このため_CEピンはGPIOにつなぐなどして，プログラムで明示的に制御しなければなりませんが，"CE Don't Care"は_CEのレベルに関係なく読み書きの操作をできるようにしてあるので，メモリ・バスに出ている_CSなどを直接つないでNANDフラッシュを制御できるようになります．

3) 専用のコントローラを介して接続

ECC回路やデータの読み書き，1ページ分の内部バッファなどを持ったコントローラに接続し，NANDの制御をこのコントローラで行う方法です．プログラムからは，コントローラのレジスタへのアクセスと，内蔵SRAMバッファへのアクセスだけで済みます．最近のアプリケーション・プロセッサは，NANDコントローラを内蔵したものがいろいろと出始めています．

● フラッシュからの読み出し

フラッシュに格納されたデータを読み出す場合には，チップのレディ状態でデータ・ポートに読み出しコマンド(`0x00`)と数バイトのアドレスを投入した後，デバイスのレディ状態を待ちます．この間にフラッシュ内部では各セルの値がリード・バッファに読み出されます．528バイトすべて読み込まれるとRY/_BYピンのレベルがレディになります．これを待ってからデータ・ポートからのデータを読み込みます．

メモリ・セルからリード・バッファへの読み出しには10～20μsかかりますが，この時間はリアルタイムOSのコンテキスト・スイッチの時間と大差ない非常に短い時間です．そのため，コンテキスト・スイッチが起こるような手法，たとえばドライバ・プログラムで時間待ちの`sleep()`を入れたり，RY/_BYを割り込みで受けるような手法は使わず，単純なbusy loopで待ち受ける実装がもっともシンプルで性能の出やすい実装方法ということになります．

● フラッシュへの書き込み

フラッシュにデータを書き込む場合は，チップのレディ状態でデータ入力コマンド(`0x80`)を投入した後，数バイトのアドレスを投入し，引き続きデータを528バイト投入します．投入したデータは内部のバッファに溜め込まれます．データ投入後に書き込みコマンド(`0x10`)を投入すると，チップ内部で書き込み動作が開始されます．書き込み動作中はRY/_BYピンがアサートされているので，GPIOポートでこれを監視するか，フラッシュにステータス・コマンドを投入して内部のステータスを読み出して監視をし続けます．データの書き込みにはおおよそ200μsかかります．リードの場合と同様にこの間をbusy loopで待つか，もしくは`sleep()`などでいったんCPU時間をほかのタスクに回すかは，システムの特性によって決まるでしょう．

● フラッシュの消去

フラッシュ上のブロックを消去する場合は，次のように操作します．チップのレディ状態で消去コマンド(`0x60`)を投入し，その後ブロック番号をアドレスとして投入，最後に消去開始コマンド(`0xD0`)を投入します．これで指定したブロックの消去が開始されます．ブロックの消去には数msかかります．消去終了は書き込みと同様にRY/_BYピンのレベルを確認するか，ステータス・リード・コマンド(`0x70`)でステータスを取得して確認します．

書き込み動作と消去動作完了後には，必ずステータス・リード・コマンド(`0x70`)を使って正常終了したかどうかを確認してください．ステータスがビジーの間のチップ・ステータスは

意味がないので無視してください．ステータスがレディになった時点で書き込みや消去の処理結果がチップ・ステータスとして示されます．これがPassならば処理完了ですが，Failになっている場合には，書き込み不良や消去不良なので，そのブロックを不良ブロックと認定して，しかるべき退避処理を実施する必要があります．消去時の不良ブロック発生については，対象ブロックに有効なデータはないので，そのブロックを不良ブロックとして管理情報を登録し，これ以降書き込みなどに使われないようにします．書き込み時の不良ブロックの場合には，そのブロックを不良ブロックとして登録するだけではなく，すでにそのブロックに正しく書き終わっているデータを，別の正常な空きブロックに書き写したうえで再度正常に終わらなかったデータの書き込み処理を再実行します．



リード/ライトのパフォーマンス

NAND型フラッシュのデータ・ポートは，通常20MHz（50ns）のリード・サイクルに対応できます．つまりフラッシュ内部のバッファへのデータの読み書きは20Mバイト/s（×8）ないし40Mバイト/s（×16）の転送スピードが出せるのです．

しかしながら，実際のシステムに接続した場合には，さまざまな要因によってこれだけのスループットを出すことは不可能です．

まず，チップ自身のオーバヘッドがあります．読み出しのセットアップ・タイムや書き込みの処理時間，さらにはコマンドとアドレスを投入するサイクルの時間などのオーバヘッドがかかります．

次にメモリ・バスがこれだけのスループットを出せません．CPUからI/Oマッピングされたデバイスへアクセスするには，アドレス・サイクルとデータ・サイクル，それにいくらかのウェイト・サイクルが必要です．さらに読み出したデータはメイン・メモリ上のバッファ領域に書き込まれるので，こちらの書き込み動作でもバスがふさがれてしまいます．

ファイル・システムなどを通じてデータをアクセスする場合は，データの転送だけでなく，ソフトウェアでECCを計算するオーバヘッドなどもあり，最大でも2～3Mバイト/sが目安になるでしょう．上記のようにメモリ・バスのサイクルを考慮して，CPUのキャッシュをぎりぎりまで使ったとしてこの倍程度が限界でしょう．これ以上のパフォーマンスが必要ならば，何か特殊なハードウェアの支援が必須です．

NANDフラッシュからのデータ転送をNANDコントローラのDMA機能を使ったとしても，同様にメモリ・バスのバンド幅がボトルネックになってくるので，データ転送の性能はそれほど向上しません．DMAを使うことでCPUを空けることができるので，システム全体の効率向上には効果があるでしょう．

以上から，データ転送速度を試算したものを表3に示します．

表3　データ転送速度の試算

フラッシュ	
ページ・サイズ	528バイト（512+16）
メモリ・セル読み出し時間	25μs
プログラミング時間	200μs
アドレス投入	4サイクル
データ・サイズ	8ビット
メモリ・バス	クロック 50MHz（20ns）4wait リード/ライト 6clock 120ns
リード	
リード・コマンド	1×120ns＝120ns
アドレス	4×120ns＝480ns
メモリ・セル読み出し	25ns
データ転送	528×120ns＝63,360ns
合計	88,960ns
転送速度	512バイト/88,960ns＝5.76Mバイト/s
ライト	
データ投入コマンド	1×120ns＝120ns
アドレス	4×120ns＝480ns
データ	528×120ns＝63,360ns
プログラム・コマンド	1×120ns＝120ns
プログラミング時間	200μs
合計	264,080ns
転送速度	512バイト/264,080＝1.94Mバイト/s



不良ブロックの処理

NAND型フラッシュは，フローティング・ゲートの絶縁部の消耗により，メモリ・セルの書き込みが規定時間で完了しなくなることがあります．その場合，消去やプログラムの実行がエラーとなるため，該当ブロックをバッド・ブロックとして以降のメモリ管理対象から除外する必要があります．

不良ブロックが発生した場合には，このブロックをこれ以降利用しないようにフラッシュの管理対象から除外します．ただし，エラーが発生した時点では，そのブロック中に有効なデータがまだ格納されたままになっている場合があります．その場合には，その有効データを正常なブロックに移動したうえで，不良ブロックを除外する必要があります．

不良ブロックは，メモリの出荷直後から，10万回以降の書き換え寿命までの間，ほぼ均等な確率で発生するといわれています．発生確率は製造プロセスやルールによっても異なりますが，フラッシュ全体でみて，カタログ・データで規定されている不良ブロックの総数を超えない範囲（おそらく一桁程度小さい）で発生します．

このような，通常使用時に発生する不良ブロックを後天性の不良ブロックと呼びます．NANDフラッシュの場合には，工場出荷時にすでに不良ブロックが存在する場合があります．これを先天性の不良ブロックと呼びます．出荷検査時点で見つかった不良ブロックにはバッド・ブロック・マークが付けられます．バッド・ブロック・マークは，通常は各ブロックの先頭ないしは先頭から規定数分のページ中の特定オフセット上のビットが

「非1」になっているかどうかで示されます．正常なブロックは消去直後のように，すべてのビットが「1」になっているので，このマークを見分けることができます．

そのため，フラッシュのドライバ・ソフトウェアやフォーマット・プログラムを開発する場合には，この点にも留意する必要があります．また，このことは，製品の機能的な側面だけでなく，工場での製品の製造工程においても意識しておく必要があることを意味します．

プリント基板上にマウントされるNAND型フラッシュに対して，初めて初期データやプログラム・イメージを書き込む際に，すでに先天的不良ブロックが存在し，そのブロックを検出する手順や，検出後にその状態を保持し，通常のシステム起動後にフラッシュのメンテナンス・プログラムにその情報が正しく渡されないといけません．

確率的にはどのブロックも同じ割合で先天性不良となりますが，実際に利用するにはこのままではたいへん不便です．そこで，NAND型フラッシュでは，工場出荷時にブロック0だけは，必ず正常ブロックであることを保証しています．

開発段階でフラッシュへのアクセスを確認したり，製品の組み立て工程で，製品情報の重要なデータを必ずここに書くなど，ブロック0の存在はいろいろと役に立ちます．

ビット・エラーとECC

NAND型フラッシュには，不良ブロックの発生のほかに，ビット・エラーという厄介な特性があります．NOR型フラッシュでは通常の使用時にビット・エラーを気にすることはほとんどありませんが，NAND型フラッシュはその構造上メモリ・セルの値を読み出すビット・ラインが複数のセルのソースとドレインの連鎖によって形成されており，専用の配線をもっていません．そのため，セルの面積を最小にでき，集積度を高めることができます．しかし，その利点の一方で，連鎖上の別のセルの状態に影響され，読み出したいセルの値が正しく読み出せない場合があるのです．これがビット・エラーとして現われます．

NOR型のフラッシュは，各メモリ・セルの値を読み出すためのラインをそれぞれもっており，ほかのセルの状態に影響を受けにくい構造になっています．

現在のNAND型フラッシュでは，ビット・エラーが出現する確率はかなり低く，エラー訂正をしないままでもデータが極端に壊れるようなことはありません．しかし，完全にエラーがないとはいえないので，これに備えてECC（Error Correction Code）を使用します．一般的にいって，もっとも簡単なECCはスマートメディアで使用されているようなハミング・コードを使用したものです．このECCは1ビット訂正，2ビット・ランダム・エラー検出の能力があります．このECCは32ビットのマイクロプロセッサで計算させても実用になる程度の計算で済みます．

多ビットのエラーを訂正できるように，無線通信やCD，ビデオなどに使われるリード・ソロモン符号を使ったさらに強力なECCを適応する場合もありますが，ソフトウェアで処理するには計算が重いため実用になりません．NANDコントローラなどの専用のハードウェア上で実装される場合がほとんどです．

ハミング・コードを使ったECCについては，本誌1999年12月号の特集第5章に詳しく解説されています．

メモリ・マッピング

フラッシュはHDDなどと異なり，直接上書き操作のできないデバイスです．あるデータの一部を書き換えるためには，そのデータの格納された物理ページを読み出し，変更箇所のデータを更新した後，新しい物理ページに書き込みます．データの格納場所が次々と物理ページ上を移動していきます．そのために，フラッシュ上の物理アドレス空間と論理アドレス空間のアドレス・マッピングを行う必要があります．データ更新をするたびにこのマッピング情報を更新して，変更箇所の論理ページ・アドレスに対応する物理ページ・アドレスを移していきます．

このような物理ページ・アドレスと論理ページ・アドレスのマッピングのメカニズムは，マイクロプロセッサのMMUが行っている仮想アドレスと物理アドレスのマッピングの考え方とそれほど大きな違いはありません．ただし，フラッシュの場合にはMMUに相当するハードウェアが仲介することが少ないため，あまり込み入ったメカニズムを実現することは実用上難しいことが多いといえます．

フラッシュは不揮発デバイスなので，システムの電源断と関わりなく継続的にメモリ・マッピング情報を保持しておかねばなりません．つまりメモリのマッピングの管理情報自身もフラッシュ上に格納する必要があるということです．この点がフラッシュの管理を難しくさせている一面です．

リクラメーション

リクラメーション（Reclamation）とは，あまり耳慣れないことばですが，日本語で「矯正」，「教化」，「再生利用」などと訳します．ここでは，上書きのできないフラッシュを管理するソフトウェアにおいて，「1度使用した箇所を，再度利用可能にするための処理」のことを「リクラメーション」と呼びます．

NAND型フラッシュにおけるリクラメーションとは，メモリ上にデータを格納するにあたり，使用済み領域を再生させる，つまり再度書き込み可能な状態に初期化する処理のことです．

具体的なリクラメーションの方法としては，プログラム側でどのページが有効かを記憶しておく必要があります．もしブロック内に有効ページと無効ページの両方がある場合は，有効ページを他ブロックへ移動してから，ブロック消去を行うという手順を踏みます．

リクラメーションの場合の処理時間について最小と最大の二つのケースを考えてみましょう．最小の場合は，対象ブロック中に有効ページがない場合なので，ブロック消去時間＋プログラム・オーバヘッド（対象ブロックに有効ページがないことの確認処理）となり，最大の場合は，対象ブロック中の全ページが有効ページの場合なので，32ページ分のフラッシュへの書き込み（A）＋ブロック消去時間（B）＋プログラム・オーバヘッド（対象ブロック中のすべてのページが有効ページだと判断する処理，移動先ページの選定時間など）となります．また，プログラムにより有効ページが少ないブロックからリクラメーションを行うようにしておけば，より実用的です．

ウェア・レベリング

フラッシュはデバイスの仕様として書き換えに対して一定の不良ブロック数の範囲内で10万回という回数の上限があります．

言い換えると，10万回の書き換えに対して，ブロック不良率は約2%以下であることをデバイス・メーカが保証していることになります．

通常の設計では，フラッシュを搭載した機器自身の製品寿命とフラッシュに対する書き換え頻度を考慮してシステムのストレージ・アーキテクチャを組み立てます．

そこで用いられるウェア・レベリング（図4）とは，フラッシュの書き換え操作が特定のブロックに集中して，そのブロックが極端に消耗して寿命が尽きることを避けるために実施される処理です．もっともオーソドックスな方法は，ブロックごとの書き換え回数を管理して，書き換え対象のブロックを選択する時点で，カウンタの値の少ないブロックを選び出すやり方です．ウェア・レベリング機能を組み込むことで，フラッシュ全体の寿命を少しでも伸ばすことができると考えられています．

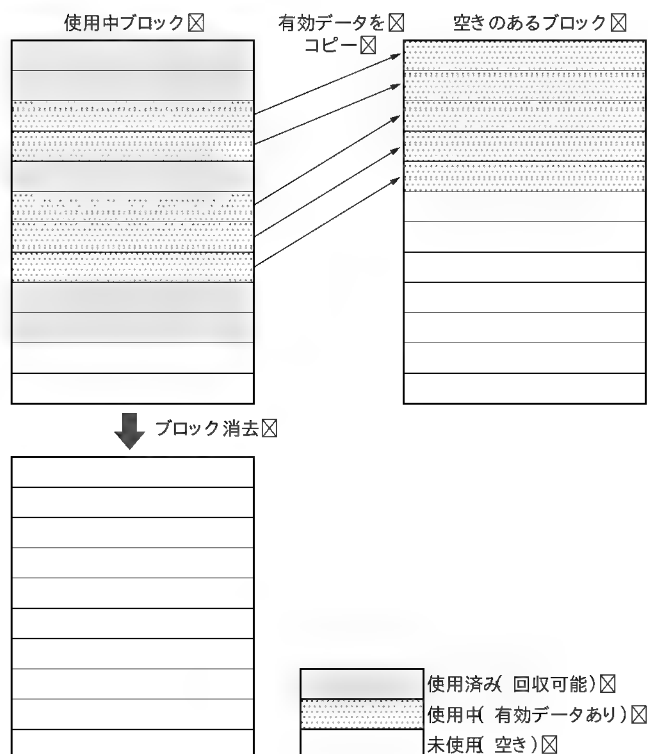


図4　ウェア・レベリングの手順

ウェア・レベリングのアルゴリズムを考えるときに，どこまで積極的に書き換え回数の平均化を実現するかの戦略がいくつかあります．

1）積極的に書き換え回数の少ないブロックを選び出す
2）特定ブロックに書き換えが集中しない防衛的方法
3）ブロック書き換えの局所集中許容

1番目の方式は，フラッシュの全ブロックの中からもっとも書き換え回数の少ないブロックを選び出して，次の書き換え対象ブロックの候補にする方法です．実際のシステムでもっとも書き換え回数の少ないブロックとは，工場出荷時に書かれたデータが変更なく保持されているブロックです．このようなブロックはCold Blockと呼ばれます．Cold Blockに書かれたデータは書き換えられる可能性が少ないわけなので，ある程度書き換え回数が多くなってしまったブロックに移しておけば，元の書き換え回数の少ないブロックが次の書き換えに使えるようになります．ただし，Cold Blockの置き換え操作では余分な空きブロックは発生しないので，この操作と同時に，使用済みブロックの回収を実施しておかないと管理情報の更新分だけ残容量を減らしてしまうだけとなってしまいます．

書き換え回数の平均化という結果を得るために，丸ごと1ブロック分のデータ書き換えが余分に発生するので，システムのオーバヘッドが高くなってしまいます．

2番目の方式は，Cold Blockには手を触れず，あくまでも使用済み領域や空きのあるブロックのみを書き換え候補の対象とします．余分なデータ転送が存在しないため，シンプルで妥当なパフォーマンスが得られます．しかし平均化の効果は限定的で，システムのユースケースによっては，局所的に書き換えが集中してしまう可能性が残ります．

3番目の方法は，ウェア・レベリングを行わないことです．ウェア・レベリングを行わなず，書き換えが局所集中するワースト・ケースを考えてみましょう．いちばん極端なパターンは，単一のブロックにのみ書き換えが行われる場合です．ウェア・レベリングは行わなくても，使用済み領域の回収（リクラメーション/ガーベジ・コレクション）は必須なので，実際のワースト・ケースでは二つのブロックを交互に書き換えることになります．

1ブロックは32ページ，1ページあたり512バイトで構成されるため，二つのブロックで32Kバイトが基本容量です．このブロックを10万回書き換えたとすると，32Kバイト×100,000＝3,200,000,000バイト＝約3.2Gバイト分の書き換えを行うことになります．わずか二つのブロックでもこれだけの容量を受け止めることができるのです．書き換え頻度についても

試算してみましょう．2ブロックで10万回なので，合計20万回の書き換えとなります．製品寿命を5年(365×5=1825日)とすると，100,000/1825≒54，1日あたり54回(54×16Kバイト=864Kバイト)の書き換えが可能です．この水準はシステムのユースケース次第では十分許容できる場合もあるでしょう．

フラッシュ・ファイル・システムと電源障害耐性

組み込みシステムにおいてもっとも忌み嫌われるのは，応答時間が読めないことです．たとえシステムにとって最悪の状況に遭遇したとしても，一定時間で応答が帰ってくることがわかっていれば，以後の対策を施すプログラムを実行することができます．しかし，いつ制御が帰ってくるかわからないのでは，事前に対応策を組み込んでおくこともできません．

「やってみなければわからない」で許されるのは，試作研究の領域であり，最終商品に搭載されるプログラムにおいては，やれば必ずこうなる，少なくともここまではがんばってそれ以上はあきらめる，といった具体的な線が把握できないとシステム設計として許容できません．

筆者の会社で開発したフラッシュ・ファイル・システムFugue(フーガ)の設計においては，応答時間もさることながら，瞬断発生時にどこまでデータの損傷が発生するのかを事前に提示できることを重要なポイントとしました．つまり，電源断が起きた場合に，どこまでを保証し，何をあきらめるのかを明確に上位プログラムに提示できることを必須要件にしたのです．

そのためには，見かけ上の高速化を演出するための内部キャッシュを排除したり，フラッシュへの書き込み処理を吟味して不確定状態のメモリ・ブロックを発生させないといったさまざまなくふうを施しました．

また，ファイル・システムを構成する各モジュールの責任分担を明確にして，フラッシュ・ドライバからファイル・システムまで各階層において，それぞれが遵守すべき項目を定めたために各層のモジュールの独立性が向上し，信頼性も高くなりました．

FATファイル・システムの電源障害耐性

では，これまでの経験を実際のFATファイル・システムに適用した場合を説明します．

FAT(File Allocation Table)は，マイクロソフト社の標準ファイル・システムの一つであり，組み込みシステムではたいへん広く使われています．記憶デバイスの連続した複数のセクタを一つのクラスタとしてとらえ，クラスタ単位でアクセスします．クラスタそれぞれにユニークな番号が割り振られています．

図5にFATファイル・システムのデータ配置の例を示します．一つのクラスタに複数のディレクトリ・エントリが配置されています．ディレクトリ・エントリに，ファイル名称，リンク情報(開始クラスタ番号)，ファイル・サイズなどが格納されます．図5において，`¥DIR1¥FILE1`へのアクセスは以下の通りです．

- まずルート・ディレクトリの領域から`DIR1`のディレクトリ・エントリを探し出します
- `DIR1`ディレクトリ・エントリに格納されているリンク情報(#10)に基づき，クラスタ#10から`FILE1`のディレクトリ・エントリを探し出します．
- `FILE1`ディレクトリ・エントリには，リンク情報として#100が格納されています．クラスタ#100には，`FILE1`のデータの一部が格納されています．図5に示している例では，これのほかにもクラスタ#102にデータが書かれています．クラスタ#102が#100につながっていることは，FATと呼ばれるテーブルに記録されています．FATエントリ#100には102が書かれています．この例では，クラスタ#102以降にデータが存在しないため，ENDマークがFATエントリ#102に記録されています

電源瞬断による破損

ファイル・システムの変更操作はおおむね下記のとおりです．

- ファイル，ディレクトリ作成
- ファイル，ディレクトリ削除
- ファイル追加書き込み
- ファイル上書き書き込み

それぞれの場合について，電源瞬断による破損の内容を見てみましょう．

● ファイル/ディレクトリの作成

作成処理は，親ディレクトリのクラスタに新たなディレクトリ・エントリを追加することです．以下の場合が考えられます．

- 親ディレクトリのクラスタにまだ未使用または再利用可能なディレクトリ・エントリがある場合は，そのディレクトリに

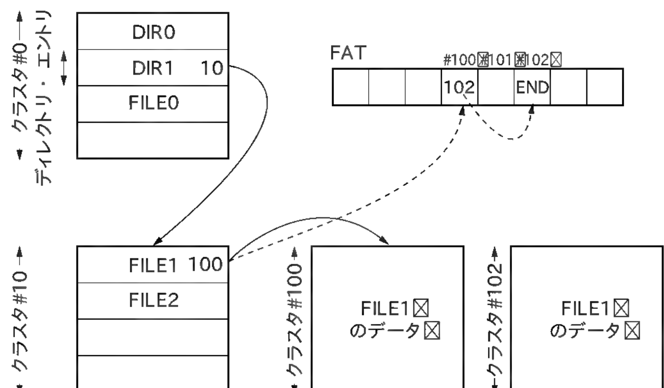


図5　FATファイル・システムのデータ配置の例

ファイル名などのデータを書き込むだけで処理が完了します．この場合，クラスタのみを更新すればよいのです．

- 親ディレクトリのクラスタに使用できるディレクトリ・エントリがない場合は，新たにクラスタを設けなければならないので，クラスタの新規書き込みおよびFATの更新が必要になります．
- 新設するディレクトリ・エントリがロング・エントリ(ファイル名称が長く32バイトのディレクトリ・エントリに収まらない場合，連続したディレクトリ・エントリを使ってファイル名称を格納するためのもの)で，親ディレクトリのクラスタをまたがってしまう場合は，クラスタ更新(または新設)が2回必要です．さらに，クラスタ新設が必要な場合は，FATも更新しなければなりません．

このように，ファイルやディレクトリを作成するとき，NAND型フラッシュへの書き込みが複数回にわたって実行されることがわかります．フラッシュのドライバが，一回の書き込みに対して，耐電源障害性を保証するだけでは，複数回の書き込みの間に電源瞬断が発生すると，論理構造が崩れる可能性があります．

● ファイル/ディレクトリの削除

ファイルやディレクトリ削除の場合は，ディレクトリ・エントリに削除マークを入れることのみで済みます．したがって，通常の場合，所属するクラスタの1回のみの更新で処理が完了します．ただし，たとえば一度にたくさんファイルを作成し，親ディレクトリのクラスタ数が二つ以上になったとき，ファイルを削除処理においてディレクトリ・エントリに削除マークを入れるだけでは，すべてのディレクトリ・エントリが削除されたクラスタがむだな領域となってしまう恐れがあります．このようなクラスタを回収する処理は，FAT更新処理を含むため，後述するように，書き込み単位でのデータの安全性を保障するドライバでは耐電源障害性を保つことが難しいのです．

● データ追加書き込み

この処理は，ファイルにデータを書き込むことです．以下の場合が考えられます．

- 書き込みデータ・サイズが十分小さく，新たにクラスタを用意しなくてもよい場合は，データ・クラスタの更新とディレクトリ・エントリ内に書かれているサイズ情報を更新しなければなりません．書き込みが複数回に渡るので，電源の瞬断でファイル・システムがダメージを受ける可能性があります．
- 新たなクラスタを用意しなければならない場合は，クラスタ更新，ディレクトリ・エントリ更新にFATの更新も必要になります．ここもやはり，書き込みが複数回にわたるため，電源の瞬断への配慮が必要です．

● データ上書き

この処理は，ファイルの既存データに新たなデータを上書きすることです．上書きであるため，FAT更新も，ディレクトリ・エントリ更新も必要ないのですが，電源瞬断への配慮が必要です．なぜなら，上書き範囲がクラスタをまたがる場合は，更新が複数回に渡ることになり，すべてのクラスタ更新が完了しないうちに電源瞬断が発生すると，それまで更新されたクラスタは新しいデータを，更新されていないクラスタは古いデータを持つことになり，いわゆる新旧データの混合状態になってしまいます．

● FAT更新

FATを更新するときに，更新するエントリの数が2以上で，フラッシュへの書き込み回数が複数回にわたると，更新が完了するまでの電源の瞬断でFATチェーンが切れてしまう可能性があります．

上述のように，FATファイル・システムにおいて，処理内容によって，フラッシュへの書き込みが複数回にわたることになり，途中での電源の瞬断でファイル・システムの論理構造が崩れてしまいます．つぎに，これらの場合への電源瞬断対策について述べます．



電源瞬断対策

電源瞬断への対策として，以下の二つの方法を考えます．

- フラッシュ・ドライバに，クラスタ単位ではなく，一連の書き込みに対してもデータの健全性を保障する機能を盛り込む方法
- クラスタ単位書き込みのみでデータ健全性を保障し，ファイル・システムがダメージを受けても，破壊されないように更新手順を配慮する方法

● 一連の書き込み保障方法

この方法では，フラッシュ・ドライバが通常の書き込み処理機能以外に，それまでの書き込みを確定するAPIを提供します．この確定APIが呼ばれる前に電源瞬断が発生しても，それまでの書き込みは単なるゴミ扱いとされ，ファイル・システムが影響を受けません．フラッシュ・ドライバが，このAPIの実行中に電源瞬断が発生しても新旧の状態が識別できるように実装されれば，ファイル・システムの耐電源障害性が保障されます．

● クラスタ単位書き込み保障方法

この方法では，ドライバがクラスタ書き込み単位でのみデータの健全性を保障します．

この場合，既述のようにファイル・システム変更操作は，記憶メディアに対して複数回にわたって書き込みを行うため，処理中に電源瞬断が発生すると，論理構造が崩れてしまいます．以降，論理構造が崩れても，ファイル・システムへのダメージを最小限にする方法を紹介します．

▶ファイル/ディレクトリの作成の場合

- 新たなクラスタを追加する必要がない場合は，一つのクラスタ更新で済むため，特別な配慮が不要です．
- 新たなクラスタを追加する必要がある場合は，書き込み順番は以下のようにします．

  ① 空きクラスタにディレクトリ・エントリを作成

このタイミングで電源瞬断が発生しても，空きクラスタがゴミになるだけで，ファイル・システムへの影響はありません．

② FATチェーンを更新する．ただしチェーンの逆順に更新する

たとえば既存のFATチェーンは，#10→ENDで，新たにクラスタ#11をつなごうとした場合，まず，FATエントリ#11にENDマーク書き込んだのち，エントリ#10に11の数字を書き込みます．かりに途中で電源瞬断が発生しても，チェーンの後半がゴミになるだけで，ファイル・システムには影響がおよびません．

▶ファイル/ディレクトリの削除の場合

- 使用済みディレクトリ・エントリのみが入っているクラスタが存在しない場合は，FAT更新が生じず，ディレクトリ・エントリの更新が一回で済むので，電源瞬断への特別な配慮が不要です．
- 使用済みクラスタが出てきた場合は，以下の手順でそのクラスタを回収します．

既存のFATチェーンが，#10→#11→ENDの例を考えます．今，クラスタ#11にあるディレクトリ・エントリがすべて削除され，これを回収したい場合，

① FATエントリ#10にENDマークを格納．ここで電源瞬断が発生しても，更新前の状態であるか，FATエントリ#11が宙ぶらりんになる状態です．どれにしても，ファイル・システムへの影響はありません．

▶データ追加書き込みの場合

- 新たにクラスタを追加しなくてもよい場合は，末尾のクラスタ更新と，ディレクトリ・エントリの更新（ファイル・サイズが変わっているので，それを更新しなければならない）が必要です．更新順番は以下のようにします．
  ① 末尾のデータ更新．ここで，電源瞬断が発生しても，ファイル・サイズがまだ古いままのため，後の書き込みなどで，書かれた部分を無効化することができます．
  ② ディレクトリ・エントリの更新．ここで，電源瞬断が発生しても，①の状態か更新完了の状態であるため，ファイル・システムがダメージを受けたとしても，①で述べたように修復可能です．
- 新たなクラスタを用意しなければならない場合は，クラスタ更新，ディレクトリ・エントリ更新にFATの更新も必要になります．更新順番は以下のようにします．
  ① クラスタにデータを書き込みます．ここで電源障害が発生してもクラスタがゴミになるだけで，ファイル・システムへの影響はありません．
  ② FATチェーンの逆順に従って，FATエントリを更新．既存FATチェーンにリンクする前に電源瞬断が発生した場合は，ファイル・システムへの影響はまったくありません．また，リンク完了した後に落ちた場合，ディレクトリ・エントリにファイル・サイズ情報がまだ残っているため，後で修復できます．
  ③ ディレクトリ・エントリを更新（ファイル・サイズ）．更新完了する前に落ちても古いサイズが残っているので，修復可能です．

▶データ上書きの場合

- クラスタをまたがない上書きは，該当クラスタを更新するだけで処理が済むため，電源瞬断への配慮が不要です．
- クラスタをまたぐ場合は，古いクラスタをそのまま更新するのではなく，新たなクラスタとFATチェーンを作成し，それができた時点で，旧FATチェーンにつなぎ換えます．

つなぎ換え処理が完了する前に電源瞬断が発生しても旧FATチェーンおよび旧クラスタが何も変更されていないので，ファイル・システムには影響がありません．つなぎ換え処理が完了したら，自動的に新しいクラスタも有効になるので，旧/新データは混じってしまうことはありません．

▶FAT更新の場合

FATチェーンを伸ばすときはチェーンの逆方向に，縮むときは順方向に行えばファイル・システムに影響が及ぶことはありません．

このように，クラスタ書き込み単位でしかデータの安全性を保障しないシステムにおいても，ファイル・システムのがんばりで何とか耐電源障害性を実現することができます．

おわりに

以上から，NAND型フラッシュをデータ・ストレージとして使用するためには，これらの要件を満たしたドライバ，ミドルウェアを用意することが必須であることが理解できると思います．

（株）京都ソフトウェアリサーチは，市場のニーズに対応する形で，フラッシュ・ファイル・システムFugueを育ててきました．NAND型フラッシュは今後さらに大容量化し，高機能化していくことはまちがいありません．市場に投入される新しいストレージ・デバイスに対してつねに高い電源障害耐性のあるファイル・システムが必要であると考えています．

ながさわ・つねや（株）京都ソフトウェアリサーチ

オープンソースのITRON仕様OS

TOPPERSで学ぶRTOS技術

第10回 ターゲットへのTOPPERSの移植

邑中 雅樹

前回は，開発環境に関する一般的な注意事項を解説し，今回の移植ターゲットを選定し，最低限のセットアップを行いました．今回は，やっと移植に入る…と予定していたのですが，あと少し準備が必要です．

まずは，XScale評価キット添付のGCCがもつ問題を解決し，補助ツールの導入を行います．その後，JSP1.4のソース・ツリーを用いた作業を始めます．

ツール整備の続き

● GCCの問題

まず，前回の作業に対する修正です．XScale評価キットにGCCが付属していることで安心しきっていましたが，評価キット付属のGCCはバージョンが古く，JSP1.4のビルド中にエラーが発生してしまいます(図1)．かなり古いバージョンなので情報もほぼ皆無で，回避方法が見つかりませんでした．そこで，別途GCCのバイナリを用意しました．近日中にInterface誌のWebサイトからダウンロード可能になる予定です．読者の皆さんはダウンロードしてお使いください[注1]．インストール方法については，誌幅のつごう上割愛します．バイナリに添付するドキュメントを参照してください．

● 補助ツールの準備

TOPPERSカーネルは，GNUのツール群を標準開発環境に据えていることから想像できるとおり，UNIX系OSでの開発を前提としています．たとえば，ビルド環境を構築するためのconfigureスクリプトはPerlで書かれており，`Makefile`には`rm`などのUNIX系ツールを前提とした記述が見られます．

TOPPERS対応をうたうボードであれば，これらのツールも含まれていると思いますが，今回のようにTOPPERS対応でないボードの場合には，自前で調達することになります．

2004年9月現在のTOPPERSプロジェクトの公式見解ではWindows環境ではCygwinの使用を推奨しています．最近のCygwinは独自インストーラの採用によって，数年前よりはインストールが簡単になりました．しかし，それでも必要なパッケージを適切にインストールするには，UNIXのツールに関する知識がいります．μClinuxなどの普及でUNIXツール類に精通している読者も多いと思いますが，UNIXツールの導入をTOPPERSカーネルの敷居の高さとして挙げる人が多いのもまた事実のようです．

今回はボードへの移植が本題なので，環境構築の細かい部分については深入りせず，バイナリ・ディストリビューションで解決することにします．

2004年9月現在，ボード添付などの形ではなく，特定のCPUアーキテクチャに特化しておらず，ソフトウェアだけ入手できるTOPPERSに特化したパッケージとして，PizzaFactory2とeFOSiがあります．eFOSiはアドバンスド・データ・コントロールズ(株)から配布されている環境です．完成度は高く，無償で配布されているのですが，GHS開発環境を前提としています．GHSの開発環境は，気軽に試してみるというわけにはいかない価格です．

そこで，今回は，PizzaFactory2の無償配布版であるOvenless Editionを使用します．PizzaFactory2については，コラム1を参照してください．

PizzaFactory2のインストールは非常に簡単です(図2)．一般的なWindowsアプリケーションと同様に，実行形式のインストーラを立ち上げてウイザードの指示に従うだけで必要な

図1 評価キット付属GCCのエラーメッセージ

```
$ make
arm-elf-gcc -c   -Wa,--gstabs -DKZ_ARM9EX -mlittle-endian    -g -O2   -DGDB_STUB
 -I. -I../../../TOPPERS/JSP/1.4/include  -I../../../TOPPERS/JSP/1.4/config/armv4
/excalibur -I../../../TOPPERS/JSP/1.4/config/armv4  -I../../../TOPPERS/JSP/1.4/k
ernel ../../../TOPPERS/JSP/1.4/kernel/startup.c
C:/DOCUME~1/MONAKA~1.MON/LOCALS~1/Temp¥ccmGk8Mj.s: Assembler messages:
C:/DOCUME~1/MONAKA~1.MON/LOCALS~1/Temp¥ccmGk8Mj.s:79: Error: <psr> expected
```

注1：先月号の「可能な限りボード添付のバイナリを使う」という原則を破ったわけになる．結果，筆者は，安定したGCCとbinutilsの組み合わせを求めてビルドを何度か繰り返す羽目に陥った．なお，ダウンロードが行えるURLは`http://www.cqpub.co.jp/interface/download/`

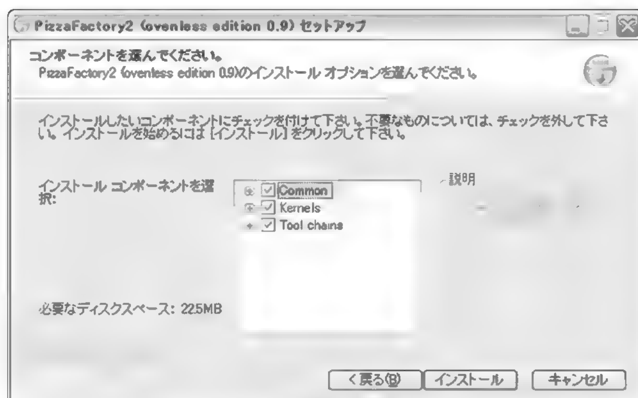


図2　PizzaFactory2のインストーラ



図3　ビルドを行った後のようす

ツールがハードディスク上に展開されます．クロスコンパイラ付きの有償版のようなプロダクトIDの入力も，無償版のOvenless Editionにはありません．

PizzaFactory2には，JSP1.4が含まれているので，改めてJSPカーネルをダウンロードしてくる必要はありません．また，cfgやchkといったJSPカーネルの公式配布版にはソース・コードしかない(開発者が独自にセルフ・ビルドしなければならない)ものも，バイナリの形式で含まれています．

動作確認

一通りのツールのインストールが終わったところで，JSP1.4のパッケージに入っているターゲットのビルドを行い，ツール類が問題なく導入できていることを確認してみます．

● シェルの立ち上げ

まずは，コマンド・ライン・シェルを立ち上げます[注2]．スタート・メニューから，[すべてのプログラム]-[MonamiSoftware]-[PizzaFactory2]-[PizzaFactory2 CommandLine]とたどってシェルを起動します．このシェルはWindowsでお馴染みのcommand.comやcmd.exeではなく，UNIXでよく用いられるbashになっています．

UNIXツールですが，Windowsの世界から隔絶されているわけではありません．たとえば，notepad.exeを実行すれば，メモ帳が立ち上がります．

● カーネルのビルド

JSP1.4には，指定したCPUとターゲット・ボードからファイル類を生成するセットアップ・スクリプトが含まれています[注3]．

```
../../TOPPERS/JSP/1.4/configure -C armv4
                                 -S excalibur
```

すると，sample1アプリケーションの生成に必要なファイル類が生成されます．続いて依存関係を解決します．

```
make depend
```

JSPに限らずTOPPERSカーネルは複数のファイルからなりたっているため，依存関係を事前に解決しておくことで再ビルドの時間を短縮することができます．最後にビルドを行います．

```
make
```

しばらくして，図3のような出力が得られてコマンド・プロンプトが戻ってくれば，ツールの導入は成功と考えてよいでしょう．

最後にビルドに使ったファイルを全消去しておきましょう．

```
rm -f *
```

これで導入直後の状態に戻りました．

● ツールの導入，まとめ

さて，これでツールの整備は完了しました．

前回も説明したとおり，ボードに添付の環境があってTOPPERSカーネルの開発に十分なツールがそろうならば，あえてサード・パーティの開発環境を使わなくてもかまいません．今回はいろいろと追加作業が発生してしまいましたが，今後，TOPPERSカーネルをバンドルするボードが増えていくに従っ

コラム1　PizzaFactory2とは?

PizzaFactory2は，Windows上でTOPPERSカーネルのクロス開発を行うために必要なUNIXツール群と独自チューニングを施したカーネル，ミドルウェア，クロスコンパイラをまとめたディストリビューション・パッケージです．

本記事で用いるOvenless Editionはオープン・ソース開発者向けの無償版で，製品版に比べて機能が大幅に削られています．とはいえ，すでにクロス開発用のGCCが手元にあってTOPPERSカーネルの開発を行いたいという条件ならば十分に活用できます．

PizzaFactory2の詳細については，

http://support.toppers-open.org/

を参照してください．

注2：シェルの起動方法は，開発環境により異なる．

注3：μITRON4.0仕様にはコンフィギュレータという概念がある．JSPのセットアップ・スクリプトがconfigureという名前なのでまぎらわしいのだが，これらには関係がないので注意が必要．ちなみにJSP1.4のカーネル・コンフィギュレータはcfg.exeである．

コラム 2 ファイル・パスの罠

UNIXツール類をWindows上で動作させるためには，いろいろめんどうなことがあります．とくに深刻なのがファイル・パス(path)の指定方法の違いです．この違いはDOSの時代からUNIX系のツールを移植するときに大きな障壁であり続けてきました．

Windows環境上でUNIX系のツールを動作させるプロジェクトは複数ありますが，2004年現在，msysとCygwinが活発に活動しています．msysもCygwinもファイル・パスを内部変換してUNIX系ツールとWindows環境との混在を実現しているのですが，変換方法には互換性がありません．また，Cygwinは，ある時点からドライブ・レターとUNIXパスとの変換方法を変えています．よって，msysとCygwinを不用意に混在させたり，場合によっては同じCygwinでも異なるバージョンを混ぜたりすると，ファイル・パスに関して不可解な挙動を示すことがあります．

さらにWindows環境に対応したUNIX由来のツールの存在が問題をややこしくしています．

たとえばPerlはその一例です．Perlにはmsysやcygwinが配布しているものに加え，ActiveState社が配布しているActivePerl，そのほか複数あり，それぞれファイル・パスの扱いが少しずつ違います．

独自の開発環境を用意しようとして失敗する例には，こういったツールの組み合わせが原因のものが少なからずあるようです．また，バイナリ・ディストリビューションを使っていたとしても，複数の開発案件を並行してこなそうとしているときに，これらの罠にはまることがありえるので注意が必要です．

本記事では，これらの問題を認識したうえでmsysベースのツールであるPizzaFactory2とCygwinベースのコンパイラという構成を取っています．可能な限り混在は避けるべきだと筆者は思いますが，どうしても混在させなければいけないときには参考になるかもしれません．

て，ツールの導入は飛躍的に容易になっていくものと思います．

読者の環境は，本記事と違う可能性はありますが，GNUベースであれば，環境構築の手順や手間が若干違ってもほぼそのまま応用できるはずです．

移植作業の実際

さて，いよいよ本題の移植作業に話題を移します．環境構築のときと同様に，移植作業で起きたことを可能な限り忠実に再現していきます．行き詰まりや失敗もこれから移植を挑戦する人にとっては重要な情報であると思うためです．

しかし，筆者と同じ失敗を繰り返す必要はありません．移植作業に挑戦する場合は，まずざっと目を通したうえで，必要な部分をつまみ食いしてください．

● ボード依存部の準備

先月号のJSPカーネルの構成で説明したとおり，機種依存部は`config`ディレクトリ以下に配置します．ソース・コードは可能な限りすでに存在するものを流用します．XScaleは厳密にはARMv5ですが，今回はすでにディレクトリがあるARMv4とみなします．また，ボード依存部は，Excalibur(アルテラ製ARM9コア内蔵FPGA搭載の京都マイクロコンピュータ製CPUボード)用の既存のコードを切り貼りして実装することにします．



図4
ボード依存部の準備

ボード依存部の名称は，ファイル名に用いることができるものなら何でもかまいませんが，英数字，マイナス(-)，アンダスコア(_)以外の文字は使わないでおくのが賢明でしょう．ここでは，`cq_pxa250eva`と命名します(図4)．

以上のことから，`config/armv4/cq_pxa250eva`フォルダを作り，`config/armv4/excalibur`フォルダの中にあるファイルをすべてコピーすればファイルの準備は完了します．

確認を兼ねて，ビルドしてみましょう．CPU依存部の指定は同じですが，ボード依存部として今決めた名前で指定します．

```
../../TOPPERS/JSP/1.4/configure -C armv4
                              -S cq_pxa250eva
```

先程の「カーネルのビルド」と同じ手順を行ってみて，エラーが出ないことを確認してみてください．

● メモリ・マップの合わせ込み

ここまでの作業でできるカーネルは，Excaliburボード上で動作するものであり，XScale評価ボードでは動作しません．

まずは，メモリ・マップをXScale評価ボード用に合わせ込んでWatchpointデバッガに読み込ませられるようにします．

ARMアーキテクチャでは，メモリの`0x00000000`番地から`0x0000001f`番地までが例外ベクタ用のアドレスとなっています．ROM化を視野に入れた多くのARM評価ボードでは，この領域はフラッシュ・メモリになっており，ブート後にRAMにバンク切り替えするような構成になっています．

この場合，例外ベクタ領域をコピーするなど煩雑な処理が必要です．Excaliburターゲット依存部は，ROM化は考えていないようで，`0x00000000`～の領域がRAMであることを前提としています．

一方，XScale評価キットのメモリ・マップは図5のようになっています．

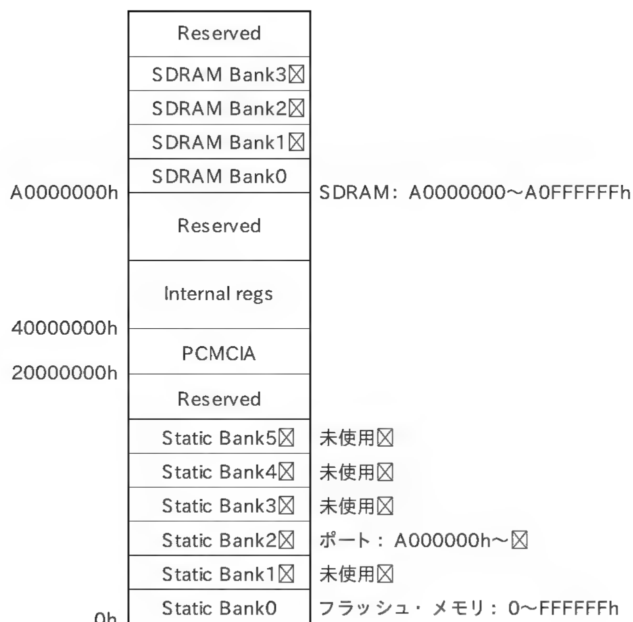


図5 XScale評価キットのメモリ・マップ

やや独特なのが例外ベクタを含む0番地付近です．0x00000000～0x00FFFFFF番地はフラッシュ・メモリ固定になっており，ROMモニタ・プログラムが書き込まれています．ROMモニタ経由でロードするプログラムは，例外ベクタを直接書き換えることができません．

しかし，例外ベクタの飛び先は0xA0000000番地から始まるアドレスに設定されています．このようなフックが存在するおかげで，アプリケーション注4からは，例外ベクタが(0x00000000番地からではなく)0xA0000000番地からの連続番地に存在するとみなせることになります．

リセット例外，未定義命令例外，FIQ割り込み例外はROMモニタが握っていますが，JSPカーネルはこれらの例外は利用しないので問題にはなりません．

今回は，これらの状況を踏まえて，Excaliburが想定する0x0000000番地からのメモリ・マップをそのまま0xa0000000番地からにずらします．

まずはカーネルのTEXTエリアとDATAエリアのスタート・アドレスを調整します．これらの情報は，config/arm4v/cq_pxa250eva/Makefile.configの後のほうにあるTEXT_START_ADDRESSやDATA_START_ADDRESSで決定されます．具体的にはを図6のようにそれぞれ書き換えます．

また，リンカ・スクリプト(excalibur.ld)も変更する必要があります．変更の方法はいろいろありますが，もっとも簡単なのは，リスト1のように，.vectorセクションの前にアドレスを明示してしまう方法です．

注4：JSPカーネルもROMモニタからみれば一種のアプリケーションといえる．

図6 アドレスの書き換え

```
TEXT_START_ADDRESS=0xA000003c
DATA_START_ADDRESS=0xA0010000
```

リスト1 リンカ・スクリプトの書き換え

```
SECTIONS
{
    . = 0xa0000000;
    .vector :
    {
        *(.vector)
    }
```

図7 リンク時に発生する可能性のあるエラー

```
start.o(.text+0x4c):../../TOPPERS/JSP/1.4/config/
                         armv4/start.S:161: relocation t
runcated to fit: R_ARM_PC24 software_init_hook
collect2: ld returned 1 exit status
```

リスト2 hookルーチン呼び出しの変更

```
        ldr  r0, =hardware_init_hook
        cmp  r0, #0x00
        bleq skip_hardware_init_hook
        ldr  pc, hardware_init_hook_k
skip_hardware_init_hook:
…
start_5:
        ldr   r0, =software_init_hook
        cmp   r0, #0x00
        bleq  skip_software_init_hook
        ldr   r0, software_init_hook_k
skip_software_init_hook:
…
hardware_init_hook_k:
        .long   hardware_init_hook
software_init_hook_k:
        .long   software_init_hook
```

binutilsのバージョン次第では，アドレスの変更にともなって，リンク時に図7のようなエラーが発生するかもしれません．

その場合には，config/armv4/start.Sをリスト2のように書き換える必要があります．

さらに，メモリ・マップとは直接の関係はありませんが，デフォルトの状態ではgdbstubを使う設定になっています．このままでは例外ベクタのための設定が無効になってしまうので，Makefile.configに以下の1行を書き加えます．

```
DBGENV :=
```

書き換えたら，すでに行ったように，configure，make depend，makeの順でカーネルをビルドします．ファイルjspができたらWatchpointデバッガ経由で早速シンボルを流し込んでみます．

● デバッグ情報の合わせ込み

Watchpointデバッガ経由でカーネルのメモリ・イメージは流し込めるはずです．

しかし，ターゲットへのダウンロードを試みると，Watchpointデバッガは，「ファイルが見つかりません」という旨のエラー・ダイアログを返してきます(図8)．

ここで無視を押すとカーネルのメモリ・イメージは流し込めます．しかし，ソース・コードの一覧が表示されるはずのプロジェクト・ウインドウには，GCCの中間生成ファイルが表示されてしまいます(図9)．この状態でもアセンブル・コード・レベルでのデバッグは行えますが，ソース・レベル・デバッグができません．

評価キットCD-ROMのドキュメントには正確な記載がなかったのですが，どうやらWatchpointデバッガは，デバッグ情報としてstabsフォーマットが渡ったときに，ELFとみなすものの，ソース・ファイルとの対応が取れなくなるようです．印刷された取扱説明書の第5章には，コンパイル時には-gdwarf-2オプションをつけるようにとの記述があります．これが先月号で取り上げたデバッグ・シンボルに関する罠です．デバッグ情報がデバッガの期待したものと異なるときに，その旨をわかりやすいエラーとして提示しないデバッガはWatchpoint以外にも複数あります．事前に知識がなければかなり悩むはずです(筆者も以前かなり悩んだ)が，対策は簡単です．stabs情報の生成をやめます．

config/armv4/cq_pxa250/Makefile.configの中でCFLAGSを設定している部分を以下のように編集します．

```
COPTS    := $(COPTS) -mcpu=arm9tdmi
```

元のCOPTSについていた-Wa，--stabsがstabsフォーマットでの生成を指示するもので，今回は邪魔なだけです．

Makefile.configの編集が終わったら，再度カーネルのビルドを行います．実は毎回configureを実行する必要はありません．rm -f *の代わりにmake cleanを実行し，次にmakeを実行してください．

カーネルのビルドが終了したら，またWatchpoint経由でターゲットにロードします．今度はプロジェクト・ウインドウに*.cや*.hといったファイルが並んでいるはずです(図10)．

● ファイル名とマクロの変更

とりあえず，デバッガで読み込めるところまではたどり着きました．ここで小休止というわけではないのですが，ボード依存部にあるファイル名を変更しておきましょう．

Excaliburボード依存部から単にコピーしてきただけの状態なので，kz_arm9ex.hやexcalibur.*など，実態と合わないファイル名になっています．JSPカーネルではターゲット依存部のファイルに命名規則は設定していません．しかし，ターゲット名同様に，2バイト文字などは避けたほうが無難でしょう．今回は表1のようにファイル名を変更しました．

また，細かいところですが，KZ_ARM9EXというマクロがconfig/armv4/cq_pxa250eva/Makefile.configに含まれています．これも，CQ_PXA250EVAという名称に変更します．

これにともない，pxa250.c，sys_defs.h，hw_timer.h，sys_support.S，sys_config.c，hw_serial.hなどでインクルード文や#ifdefの条件を編集する必要が出てきます．

実際の作業では，ファイル名を変更してしまった後でmake dependを実行し，エラーが出たところを潰していくという付け焼刃的な対応が実はいちばん楽です．

これまでの作業でMakefile.configへの変更をいくつか行いました．Makefileにはマクロ定義などが含まれるため検

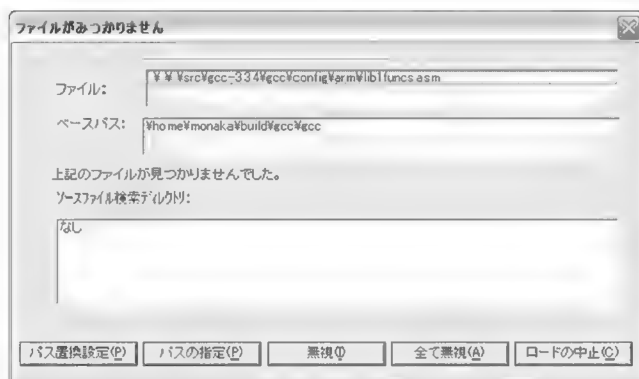


図8　エラー・ダイアログ

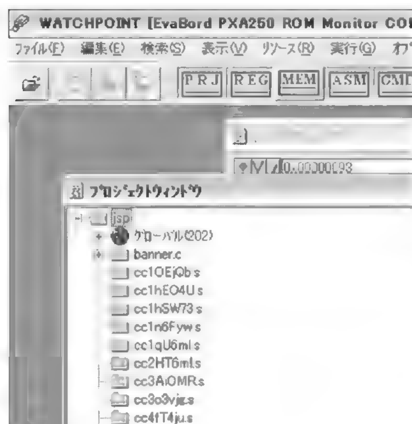


図9
中間生成ファイルが表示されている

表1
変更するファイル名

変更前	変更後
excalibur.c	pxa250.c
excalibur.h	pxa250.h
excalibur.ld	pxa250.ld
Kz_arm9ex.h	cq_pxa250eva.h



図10
期待通りのファイルが表示されている

出しにくいバグの原因になることがめずらしくありません．念のため，ここでMakefile.configの全部を，リスト 3に掲載します．

ブートさせてみる

移植のどの段階から実機でのデバッグを開始するのかについては，開発者によって主義主張があって興味深いところです．古くからの流儀に則ってソース・コード・レビュを繰り返し，正常動作を確信したうえでターゲット・デバッグへ移るべきというのが，おそらく正論でしょう．

しかし，今回は，周辺ペリフェラルの移植にもいっさい手がついていない状態にもかかわらず，まず実行してしまいます．少しずつでも動いている実感があればこそ移植作業も楽しくできようというものです，よね?

● リセット

今までの作業によって，カーネルはターゲット・ボードへインストールされました．この状態で，Watchpointのウィンドウ上にあるリセット・ボタンを押してみてください．すると，逆アセンブル・ウィンドウが立ち上がるはずです（図11）．

また，プログラム・カウンタはリセット例外ベクタに対応する0xa0000000番地を指しているはずです．この番地の命令はLDR PC, 0xa0000020になっているはずですが，そうでなければ，メモリ・マップの合わせ込みのどこかで失敗しています．問題がなさそうなら先に進みましょう．

● 実行

さて，実行です．

とはいっても，GOを押せば一気にハードウェアの初期化部まで実行し，確実に暴走します．逆アセンブラ・ウィンドウが表示されている状態で数回STEPを押してみてください．hardware_init_hookに飛ぶ寸前（0xa0000050番地辺り）くらいまでプログラム・カウンタが無事に動いたでしょうか?もし動いたならば，ここまでの作業にまちがいはありません．

まだカーネルの玄関に立ったところですが，まずはブートしました．

● カーネルの入り口まで進める

さて，ステップ実行を続けていると，hardware_init_hookの付近で暴走します．hardware_init_hookは関数名どおりハードウェアの初期化を行うルーチンで，本質的にボード依存性が高いところです．カーネルが起動する前に初期化をしておきたいDRAMコントローラなどのために存在しますが，今回はROMモニタがそのあたりをやってくれると期待できるので，ごっそり取り除いても問題ありません．

具体的にはsys_support.Sの46行目付近から248行目付近までの範囲は不要です．

シンボルまで削ったらリンク時にエラーが出るのではと心配な場合には，次回掲載するコラムを参照してください．

リスト 3　armv4/cq_pxa250eva/Makefile.config

```
#
#  @(#) $Id: Makefile.config,v 1.6 2003/12/01
06:29:15 honda Exp $
#

#
#  Makefile のシステム依存部分（CQ_PXA250EVA用）
#

#
#  コンパイル・フラグ
#
INCLUDES := $(INCLUDES) -I$(SRCDIR)/config/$(CPU)/$(SYS)
COPTS    := $(COPTS) -mcpu=arm9tdmi
LDFLAGS  := $(LDFLAGS) -mcpu=arm9tdmi  -N
DBGENV :=

#
# カーネルに関する定義
#
KERNEL_DIR      := $(KERNEL_DIR):$(SRCDIR)/config/$(CPU)/$(SYS)
KERNEL_ASMOBJS := $(KERNEL_ASMOBJS) sys_support.o
KERNEL_COBJS    := $(KERNEL_COBJS) sys_config.o pxa250.o

#
#  リンカ・スクリプトの定義
#
LDSCRIPT = $(CPU)/$(SYS)/pxa250.ld


#
#  ターゲット・ボード依存の定義
#

#  CQ_PXA250EVA用
COPTS   := $(COPTS) -DCQ_PXA250EVA -mlittle-endian
LDFLAGS := $(LDFLAGS) -mlittle-endian
TEXT_START_ADDRESS=0xa000003c
DATA_START_ADDRESS=0xa0010000
```

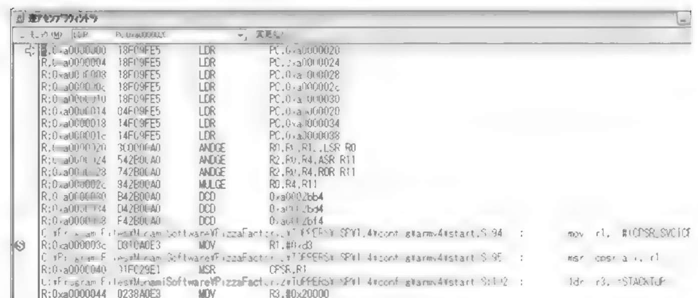


図11　逆アセンブル・ウィンドウ

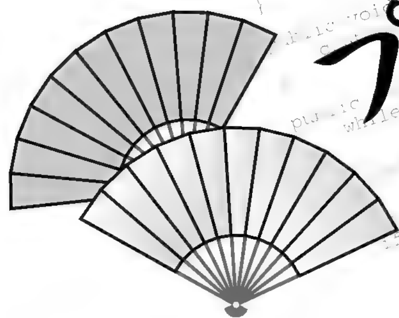
さて，再ビルドし，ボードにロードしてみましょう．先程と同様にリセットし，ステップ実行を行ってみて，bl kernel_startのところ（0xa000000094）までくることを確認できれば，ブート・ストラップ部分の移植は完了です．

今月のまとめ

今月は環境構築に誌幅を取られてしまいました．先月号でバイナリ・ディストリビューションの利用を推奨しましたが，図らずも反面教師になってしまったようです．ともあれ，ブート・ストラップまで実行できました．来月は，kernel_startの先にある，JSPカーネルの初期化部に話題を移します．

むらなか・まさき　（資）もなみソフトウェア





プログラミングの要

宮坂 電人

第18回
辞書圧縮 ――フレーズ単位の効率的な圧縮法

辞書圧縮：フレーズに注目して圧縮を行う

前回述べたとおり，圧縮の原理とは，

● 情報の偏りに着目し，それを利用して符号化する

ことです．前回紹介したランレングス符号化やハフマン符号化は一つのシンボルの出現頻度に着目して圧縮する方法でした．特定のシンボルが平等に出現せず，連続して同じシンボルが出現する情報の偏りに着目して，それを利用した符号化（ランレングス符号化）を行ったり，あるいは特定のシンボルの使用頻度がほかのシンボルの使用頻度とは違うという情報の偏りに着目した符号化（ハフマン符号化）を行うというものでした．

これらの圧縮方法よりもさらに圧縮できる方法はないかというと，実は一つのシンボルに着目するのではなく「フレーズ」に着目した圧縮方法があります．

たとえば，「TICKTACKTICKTOE」という120ビット（15バイト）のデータがあったとしましょう．このデータを観察すると「TICK」，「TACK」，「TOE」の3種類のフレーズからできていることに気づきます．ここで元のデータを，

TICK → 0
TACK → 10
TOE → 11

という2進数のビット・データに置き換えるテーブル（辞書）を用意して符号化すると「010011」に変換され，なんと120ビットあったデータが6ビットにまで圧縮されます．

ここで示したような元データから辞書にあるフレーズを発見し符号化することで圧縮する方法が「辞書圧縮」の原理です．原理はとても簡単ですが，問題はどのように辞書を構築するかです．これは簡単とはいえません．そもそもどうやって辞書を構築するのかが難しいのです．人間が直感的に「TICKTACKTICKTOE」から三つのフレーズを発見するのは雑作もないことですが，コンピュータのプログラムとして実現するのはちっとも簡単ではありません．下手をするとパターン認識や人工知能の領域にまで踏み込みそうです．

注1：復号化作業と辞書構築/辞書検索も同時に行われる．

静的辞書と動的辞書：辞書作成の二つの方法

さきほどの「TICKTACKTICKTOE」で作成した辞書は固定的なもので，あらかじめ全体のデータを走査して三つのフレーズを発見したものです．圧縮作業で使う辞書がいったん構築されると，最後まで辞書の内容が変化しない場合，その辞書を「静的辞書」と呼ぶことがあります．前回紹介したハフマン符号化もある意味，静的辞書を使った圧縮方法といえます．ただし，こちらはフレーズの代わりにシンボルを辞書に登録しただけともいえますが．

それに対して，固定的な辞書を作らず，データの読み取りと同時に辞書の構築を始め，辞書の内容がどんどん変更されていく方法もあります．この方法で作成される辞書を「動的辞書」と称します．一見，手間をかけてわざと効率の悪いことをやっているように思えます．しかし情報の偏りが局所的であった場合，動的辞書のほうがその変化にうまく対応できます．ファイルの先頭あたりで頻出するフレーズと末尾あたりで頻出するフレーズが違う場合，静的辞書だとどっちつかずの中途半端となり，圧縮効率が悪くなる可能性が大きいわけです．また動的辞書は処理が1パスになる利点があります．圧縮作業と辞書構築/辞書検索が同時に行われるため[注1]，かえって実行効率が良くなるのです．静的辞書の場合，1パス目で辞書を構築し，2パス目でようやく圧縮作業を始めるので，どうしても遅くなりがちです．

LZ77とLZ78：圧縮と辞書構築/検索を同時に

圧縮作業と辞書構築/辞書検索が同時に行われる圧縮方法は，だれもが簡単に思いつく方法とはいえません．そもそも辞書を動的に構築したり変更するということがイメージしにくいからです．1977年，Jacob Ziv氏とAbraham Lempel氏の連名で発表された圧縮方法（LZ77）は，それまでにだれも思いつかなかった画期的な方法で，これ以来，両者の頭文字を取って「LZ符号

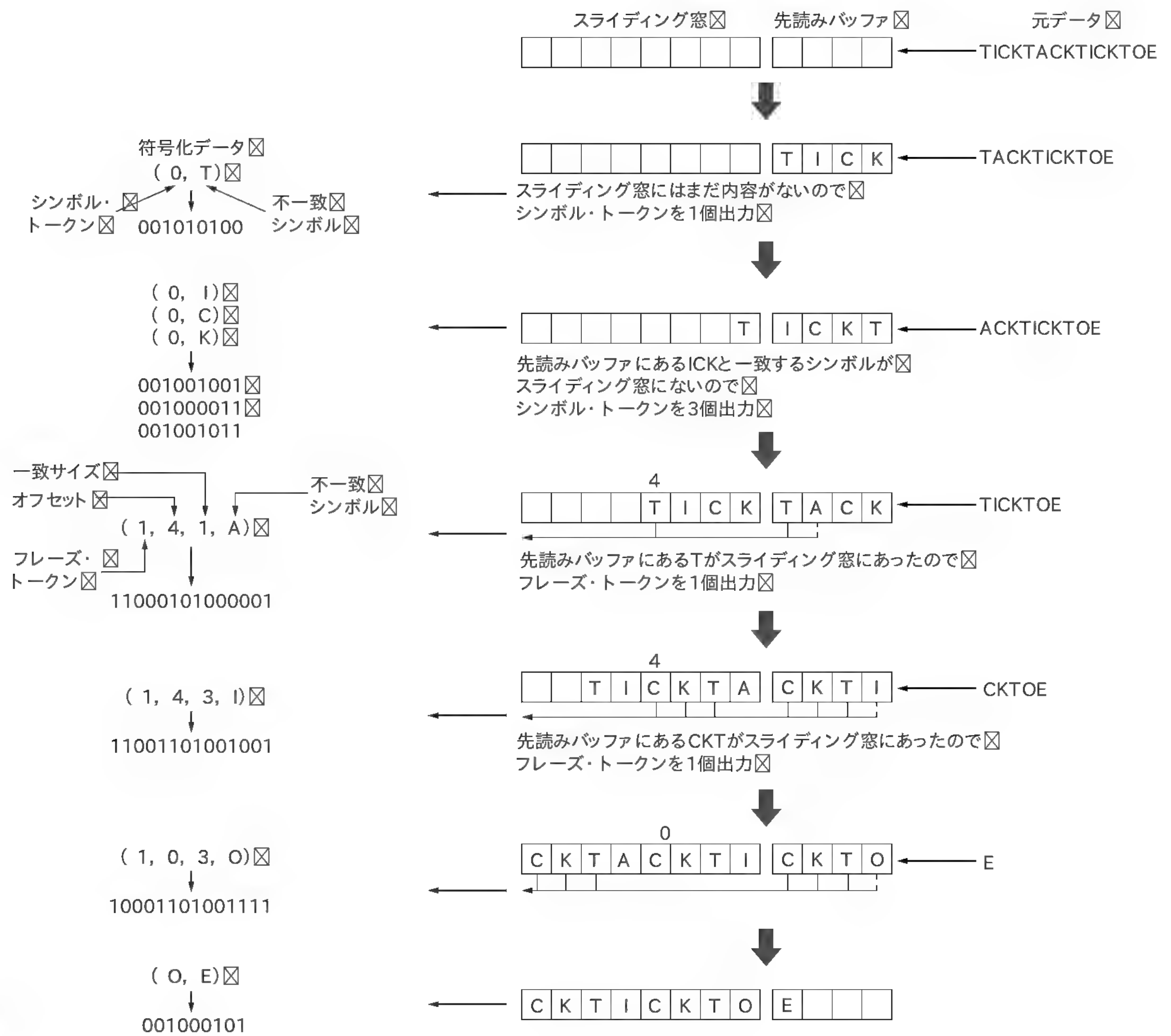


図1
スライド辞書圧縮の符号化

化」[注2]と呼ばれる方法が有名になりました．

また，翌1978年にも圧縮作業と辞書構築/辞書検索が同時に行われる別の圧縮方法(LZ78)を発表し，以来，現在に至るまで圧縮のプログラミングに関わる人たちにとってLZ符号化は絶対に無視できない存在となりました．ところでLZ77とLZ78はほぼ同時期に発表されたものの，原理はまるで違います．LZ77は「スライディング窓(sliding window)」というしかけを利用した若干トリッキーな感じがするのに対し，LZ78は動的辞書圧縮法の本来の特徴である「辞書を育てる」感じが濃厚です．よく考えると，LZ77は辞書を育てているのではなく，どんどん辞書を入れ換えている感じです．そこで，ここからはLZ77やその派生方法を「スライド辞書圧縮」と称し，LZ78やそのその派生方法を「動的辞書圧縮」と称し，両者の違いがわかるよう説明していきましょう．

注2：本来は連名の順番で「ZL符号化」と呼ばないといけないが，ある人が引用するときに「LZ」としてしまい，こっちのほうが定着してしまったらしい．

スライド辞書圧縮

スライド辞書圧縮を簡単に説明すると「スライディング窓」と「先読みバッファ(look-ahead buffer)」と呼ばれる2種類の配列を用意し，そこに圧縮したいデータを通過させる方法です．このとき先読みバッファにあるデータとまったく同じフレーズがスライディング窓にあるかないかを探し，もしあったなら，そのフレーズのスライディング窓内オフセット位置と一致サイズ，後続の1バイト(不一致シンボル)を符号化します(図1)．うまく符号化できたなら圧縮されるわけです．

スライディング窓と先読みバッファのサイズは2の累乗になるように設定します．また先読みバッファのサイズはスライディング窓のサイズと等しいか，それよりも小さくなるように設定します．かりにスライディング窓が8バイト，先読みバッファが4バイトとすると，それぞれ，2の3乗，2の2乗となるので符号化されるとき，スライディング窓内のオフセット位置

は3ビット，一致サイズは2ビット[注3]となります．スライディング窓内に一致するフレーズが見つからなかった場合，0を1ビット出力し，不一致シンボルを8ビット出力します（計9ビットで「シンボル・トークン（symbol token）」と称する）．一致するフレーズが見つかった場合，1を1ビット出力し，スライディング窓内オフセットを3ビット出力し，一致サイズを2ビット出力し，最後に不一致シンボルを8ビット出力します（計14ビットで「フレーズ・トークン（phrase token）」と称する）．

復号化するときは最初に1ビットを読み取り，これが0（シンボル・トークンの始まり）なら後続の不一致シンボルをスライディング窓に押し込み，最初の1ビットが1（フレーズ・トークンの始まり）ならスライディング窓内のオフセット位置3ビット，一致サイズ2ビットを読み取り，復元したフレーズをスライディング窓に押し込み，続けて不一致シンボルをスライディング窓に押し込みます．こうしてスライディング窓内に元データが復元されていくわけです（図2）．

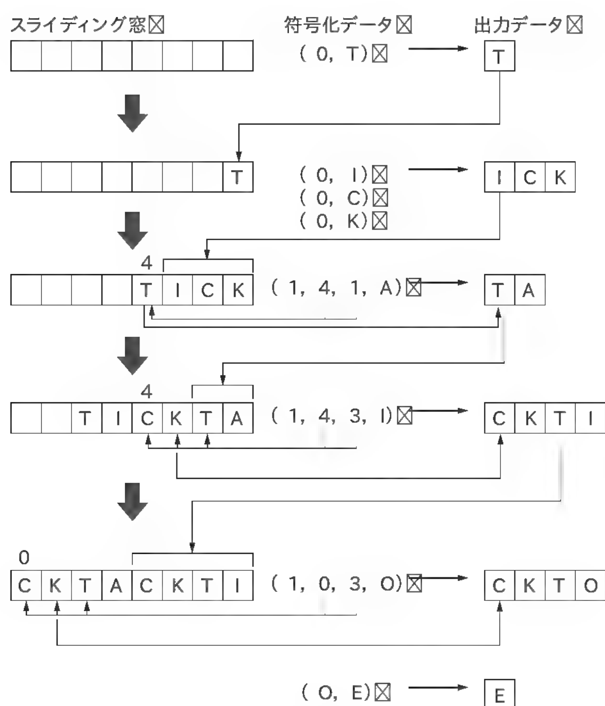


図2　スライド辞書圧縮の復号化

スライド辞書圧縮で「辞書」と称しているのはスライディング窓そのものであり，出現するフレーズの頻度に偏りがあってスライディング窓に出現頻度の高いフレーズを発見して符号化できれば圧縮率はその分，高くなるはずです．反面，フレーズの出現頻度に偏りがない場合，不一致シンボルが多くなり，シンボル・トークンだらけになるので圧縮率は期待できなくなります．

スライド辞書圧縮の実装例

前回紹介したハフマン符号化と同様，可変長ビット数で読み書きがあるため，

- ReadByteBase ——バイト単位で読み取りをするオブジェクトのベース・クラス
- WriteByteBase ——バイト単位で書き込みをするオブジェクトのベース・クラス
- ReadBitClass ——ビット単位で読み取るクラス
- WriteBitClass ——ビット単位で書き込むクラス

という4種類のクラスを利用しています．これらについては前回説明したので，詳細は前回の記事を参照してください．

最初に決めるのはスライディング窓と先読みバッファのサイズです．あまり小さいとピックアップできるフレーズが少なくなり，圧縮率が期待できません．かといって大きいとフレーズ・トークンのサイズが大きくなったり，フレーズの検索に時間がかかって不都合です．ここでは適当な数値としてスライディング窓は4096バイト（2の12乗バイト），先読みバッファは32バイト（2の5乗）とします．リスト1に今回用いたスライド辞書圧縮の定数を示します．

スライド辞書圧縮にとって最大のボトルネックは先読みバッファにあるフレーズがスライディング窓に存在するかを探す処理です．これは注意深くプログラムしないと処理速度を大幅に落とすことになります．とはいうものの，ここではプログラムを解読しやすくするため，あえて力技でコーディングした例をリスト2に示します．

符号化処理はリスト3のようになります．処理の最初で圧縮前のサイズを書き込みますが，これは復元するときの目安として必要だからです．

注3：一致サイズは先読みバッファのサイズを越えられないので．

リスト1　スライド辞書圧縮の定数

```
#define LZ_SD_TYPE_BITS_SIZE      1     //タイプ識別ビットの数
#define LZ_SD_WINDOW_BITS_SIZE    12    //スライディング窓の表現ビット数
#define LZ_SD_LOOKBUF_BITS_SIZE   5     //先読みバッファの表現ビット数
#define LZ_SD_NEXTBYTE_BITS_SIZE  8     //不一致シンボルの表現ビット数
#define LZ_SD_WINDOW_SIZE         4096  //スライディング窓のサイズ(2のLZ_SD_WINOFF_BITS_SIZE乗であること)
#define LZ_SD_LOOKBUF_SIZE        32    //先読みバッファのサイズ(2のLZ_SD_LOOKBUF_BITS_SIZE乗であること)

//フレーズ・トークンのビット数
#define LZ_SD_PHRASE_BITS_SIZE (LZ_SD_TYPE_BITS_SIZE+LZ_SD_WINDOW_BITS_SIZE+LZ_SD_LOOKBUF_BITS_SIZE+LZ_SD_NEXTBYTE_BITS_SIZE)
//シンボル・トークンのビット数
#define LZ_SD_SYMBOL_BITS_SIZE (LZ_SD_TYPE_BITS_SIZE+LZ_SD_NEXTBYTE_BITS_SIZE)
```

リスト 2　フレーズの一致箇所を探す処理

```
//スライド辞書圧縮のクラス
class LZ_SD_Compress {
    //スライディング窓
    uint_8_t mSlidingWin[LZ_SD_WINDOW_SIZE];
    //先読みバッファ
    uint_8_t mLookaheadBuff[LZ_SD_LOOKBUF_SIZE];

    //スライディング窓と先読みバッファを比較して
    // 最大一致箇所を探す
    //戻り値は最大一致数(0～),oOffsetにスライディング窓内の
    // オフセット値(0～),oNextByteに不一致シンボルをいれる
    uint_32_t compare_window(uint_32_t& oOffsetInWin,
                             uint_8_t& oNextByte) {
        uint_32_t aLongest = 0; //最大一致数
        //一致箇所がない場合の対策
        oOffsetInWin = 0;
        oNextByte = mLookaheadBuff[0];
        //スライディング窓内を力技で走査する
        for(uint_32_t aOffset = 0; aOffset <
                LZ_SD_WINDOW_SIZE; aOffset++){
            uint_32_t aMatch = compare_window_sub(aOffset);
            //一致数が最大なら各情報を更新する
            if(aMatch > aLongest){
                aLongest = aMatch;
                oOffsetInWin = aOffset;
                oNextByte = mLookaheadBuff[aMatch];
            }
        }
        return aLongest;
    }
    uint_32_t compare_window_sub(uint_32_t iOffset) {
        //スライディング窓内のオフセット
        uint_32_t aI = iOffset;
        uint_32_t aJ = 0; //先読みバッファ内のオフセット
        while(aI < LZ_SD_WINDOW_SIZE
                && aJ < LZ_SD_LOOKBUF_SIZE - 1){
            if(mSlidingWin[aI] == mLookaheadBuff[aJ]){
                ++aI;
                ++aJ;
            }else{
                return aJ;
            }
        }
        return aJ;
    }
```

リスト 3　スライド辞書圧縮の符号化処理

```
//圧縮を行う.
// 成功すれば圧縮後のサイズを返す. 失敗すれば0を返す
//iRBB=読み取りオブジェクト, iWBB=書き込みオブジェクト,
// iStartPos=読み取り開始位置(0～),iProcSize=圧縮対象のサイズ
uint_32_t compress(ReadByteBase& iRBB,WriteByteBase& iWBB,
                   uint_32_t iStartPos,int_32_t iProcSize) {
    //サイズが0以下なら戻る
    if(iProcSize <= 0)
        return 0;
    //書き込み済みサイズをリセット
    iWBB.reset_wrote_size();
    //圧縮対象サイズを書き込む
    if(!iWBB.write_4byte(iProcSize))
        return 0;
    //bit単位書き込みオブジェクトを用意する
    WriteBitClass aWBits(iWBB);
    //スライディング窓と先読みバッファをクリアする
    std::memset(mSlidingWin,0,LZ_SD_WINDOW_SIZE);
    std::memset(mLookaheadBuff,0,LZ_SD_LOOKBUF_SIZE);
    //先読みバッファにデータを読み込む
    uint_8_t* aP = mLookaheadBuff;
    uint_32_t aI = 0;
    while(aI < LZ_SD_LOOKBUF_SIZE
              && aI < static_cast<uint_32_t>(iProcSize)){
        uint_8_t aDat;
        if(!iRBB.read_1byte(aDat))
            return 0;
        *aP++ = aDat;
        ++aI;
    }
    //読み取りオブジェクトから読み取ったバイト数
    uint_32_t aReadBytes = aI;
    //符号化すべき残りバイト数
    int_32_t aRemainBytes = iProcSize;
    //処理すべき読み取りデータがある限り, 符号化を繰り返す
    while(aRemainBytes > 0){
        //書き込みトークン(フレーズ・トークンまたは
                          シンボル・トークン)
        uint_32_t aToken;
        int aTokenBits;     //書き込みトークンの表現ビット数
        //スライディング窓または先読みバッファ内のオフセット
        uint_32_t aOffset;
        uint_8_t aNextByte; //不一致シンボル
        //スライディング窓と先読みバッファを比較して
        // 一致箇所を探す
        uint_32_t aMatchLen = compare_window(aOffset,
                                             aNextByte);
        if(aMatchLen > 0){ //一致箇所があった場合
            //フレーズ・トークンを作成する
            aTokenBits = LZ_SD_PHRASE_BITS_SIZE;
            aToken = (1 << (LZ_SD_WINDOW_BITS_SIZE
                            +LZ_SD_LOOKBUF_BITS_SIZE
                            +LZ_SD_NEXTBYTE_BITS_SIZE)) |
                     (aOffset << (LZ_SD_LOOKBUF_BITS_SIZE
                            +LZ_SD_NEXTBYTE_BITS_SIZE)) |
                     (aMatchLen << LZ_SD_NEXTBYTE_BITS_SIZE) |
                         static_cast<uint_32_t>(aNextByte);
        }else{ //一致箇所がない場合
            //シンボル・トークンを作成する
            aTokenBits = LZ_SD_SYMBOL_BITS_SIZE;
            aToken = static_cast<uint_32_t>(aNextByte);
        }
        //トークンを書き込む
        if(!aWBits.write_bits(aTokenBits,aToken))
            return 0;
        //(不一致シンボルを出力するので+1して調整している)
        ++aMatchLen;
        //スライディング窓と先読みバッファの内容を移動する
        std::memmove(mSlidingWin,mSlidingWin
                  + aMatchLen,LZ_SD_WINDOW_SIZE - aMatchLen);
        std::memmove(mSlidingWin + (LZ_SD_WINDOW_SIZE
                  - aMatchLen),mLookaheadBuff,aMatchLen);
        std::memmove(mLookaheadBuff,mLookaheadBuff
                  + aMatchLen,LZ_SD_LOOKBUF_SIZE - aMatchLen);
        //先読みバッファを読み取る
        aOffset = LZ_SD_LOOKBUF_SIZE - aMatchLen;
        while(aOffset < LZ_SD_LOOKBUF_SIZE
           && aReadBytes < static_cast<uint_32_t>(iProcSize)){
            uint_8_t aDat;
            if(!iRBB.read_1byte(aDat))
                return 0;
            mLookaheadBuff[aOffset] = aDat;
            ++aOffset;
            ++aReadBytes;
        }
        //処理したバイト数だけ減らす
        aRemainBytes -= static_cast<int_32_t>(aMatchLen);
    }
    //bit単位書き込みオブジェクトをフラッシュする
    if(!aWBits.close_buffer())
        return 0;

    return iWBB.get_wrote_size();
}
```

リスト 4　スライド辞書圧縮の復号化処理

```
//展開を行う.
// 成功すれば展開後のサイズを返す，失敗すれば0を返す
//iRBB=読み取りオブジェクト，iWBB=書き込みオブジェクト，
// iStartPos=読み取り開始位置(0～)
uint_32_t uncompress(ReadByteBase& iRBB,WriteByteBase& iWBB,
                      uint_32_t iStartPos){
    //書き込み済みサイズをリセット
    iWBB.reset_wrote_size();
    //読み取り開始位置をiStartPosに移動
    if(!iRBB.rewind_ptr(iStartPos))
        return 0;
    //最初の4バイトを読む (復元サイズ)
    uint_32_t aOrigSize;
    if(!iRBB.read_4byte(aOrigSize))
        return 0;
    //復元すべき残りバイト数
    int_32_t aRemainBytes = static_cast<int_32_t>(aOrigSize);
    //スライディング窓と先読みバッファをクリアする
    std::memset(mSlidingWin,0,LZ_SD_WINDOW_SIZE);
    std::memset(mLookaheadBuff,0,LZ_SD_LOOKBUF_SIZE);
    //bit単位読み取りオブジェクトを用意する
    ReadBitClass aRBits(iRBB);
    //復元すべき読み取りデータがある限り，復元を繰り返す
    while(aRemainBytes > 0){
        //1ビットだけ読み取る
        uint_8_t aDat;
        if(!aRBits.read_1bit(aDat))
            return 0;
        uint_32_t aBufLen;   //バッファ内に送り込んだサイズ
        uint_32_t aNextByte; //不一致シンボル
        if(aDat){ //フレーズ・トークンの場合
            uint_32_t aOffset;
            //スライディング窓内オフセットを読み取る
            if(!aRBits.read_bits(LZ_SD_WINDOW_BITS_SIZE,
                                 aOffset))
                return 0;
            //一致サイズを読み取る
            if(!aRBits.read_bits(LZ_SD_LOOKBUF_BITS_SIZE,
                                 aBufLen))
                return 0;
            //不一致シンボルを読み取る
            if(!aRBits.read_bits(LZ_SD_NEXTBYTE_BITS_SIZE,
                                 aNextByte))
                return 0;
            //スライディング窓内のデータを書き込む
            uint_32_t aI = 0;
            while(aI < aBufLen && aRemainBytes > 0){
                aDat = mSlidingWin[aOffset + aI];
                if(!iWBB.write_1byte(aDat))
                    return 0;
                //先読みバッファに書き込んだ内容を書き込む
                mLookaheadBuff[aI] = aDat;
                ++aI;
                --aRemainBytes;
            }
            //不一致シンボルを書き込む
            if(aRemainBytes > 0){
                aDat = static_cast<uint_8_t>(aNextByte);
                if(!iWBB.write_1byte(aDat))
                    return 0;
                //先読みバッファに書き込んだ内容を書き込む
                mLookaheadBuff[aI] = aDat;
                --aRemainBytes;
            }
            ++aBufLen; //(不一致シンボルを含めるので)
        }else{ //シンボル・トークンの場合
            //不一致シンボルを読み取る
            if(!aRBits.read_bits(LZ_SD_NEXTBYTE_BITS_SIZE,
                                 aNextByte))
                return 0;
            //不一致シンボルを書き込む
            aDat = static_cast<uint_8_t>(aNextByte);
            if(!iWBB.write_1byte(aDat))
                return 0;
            //先読みバッファに書き込んだ内容を書き込む
            mLookaheadBuff[0] = aDat;
            --aRemainBytes;
            aBufLen = 1;
        }
        //スライディング窓の内容をシフトさせる
        std::memmove(mSlidingWin,mSlidingWin + aBufLen,
                     LZ_SD_WINDOW_SIZE - aBufLen);
        //先読みバッファの先頭データをスライディング窓につめる
        std::memmove(mSlidingWin + (LZ_SD_WINDOW_SIZE
                     - aBufLen),mLookaheadBuff,aBufLen);
    }

    return iWBB.get_wrote_size();
}
```

復号化処理はリスト4のようになります．

動的辞書圧縮：辞書を「育てる」

スライド辞書圧縮は，辞書であるスライディング窓の内容をどんどん入れ替えるという特徴があります．これはファイルの先頭あたりと末尾あたりでフレーズの出現頻度が大幅に違う場合，辞書が固定的ではないので出現頻度の変化に上手に対応できることを意味します．しかし，せっかく見つけたフレーズをどんどん捨て去っていて，もったいない(？)感じもあります．これに対し，動的辞書圧縮は見つけたフレーズをどんどん辞書に登録して「育てる」ことで，元データを読み進むにつれ，辞書に登録されたフレーズがヒットしやすくなり，どんどん圧縮率が高くなることが期待できます．

動的辞書圧縮は次のような処理をします．圧縮したいデータのうち，すでに辞書に登録されているフレーズがあるかを探し，もしあるなら，なるべく長いフレーズを見つけ，そのフレーズが登録されているインデックス番号を符号にします．と同時に，そのフレーズに続く1バイト(不一致シンボル)を付加したものを圧縮データとします．辞書にフレーズが見当たらない場合はインデックス番号を0と考え，同じく，不一致シンボル1バイトを付加したものを圧縮データとします．そして，ここからが辞書を育てるという感覚なのですが，辞書内にあったフレーズと不一致シンボルを足したフレーズを(辞書内にフレーズがなかった場合は不一致シンボルのみを)新たに辞書に登録します(図3)．こうすることでどんどん辞書が大きくなり，それにつれ辞書内でヒットするフレーズの確率も増えていき，符号化できるフレーズが頻出し，圧縮率が高くなっていきます．

復号化するときは辞書内のインデックス番号と不一致シンボル1バイトを元に復元します(図4)．おもしろいことに復号化のときも辞書を育てています．辞書の大きさは小さいものからだんだん大きいものになるので，インデックス番号を表現するビット数は固定する必然性がありません．というよりも固定すると圧縮率を悪くしたり，辞書の登録数に上限ができてしまいます．最初は辞書のサイズは0ですから，インデックス番号はどのみち0になるはずです．したがって，最初に符号化するときはインデックス番号を省略して，不一致シンボルのみにできます．符号化と復号化のいずれにおいても，1フレーズを処理

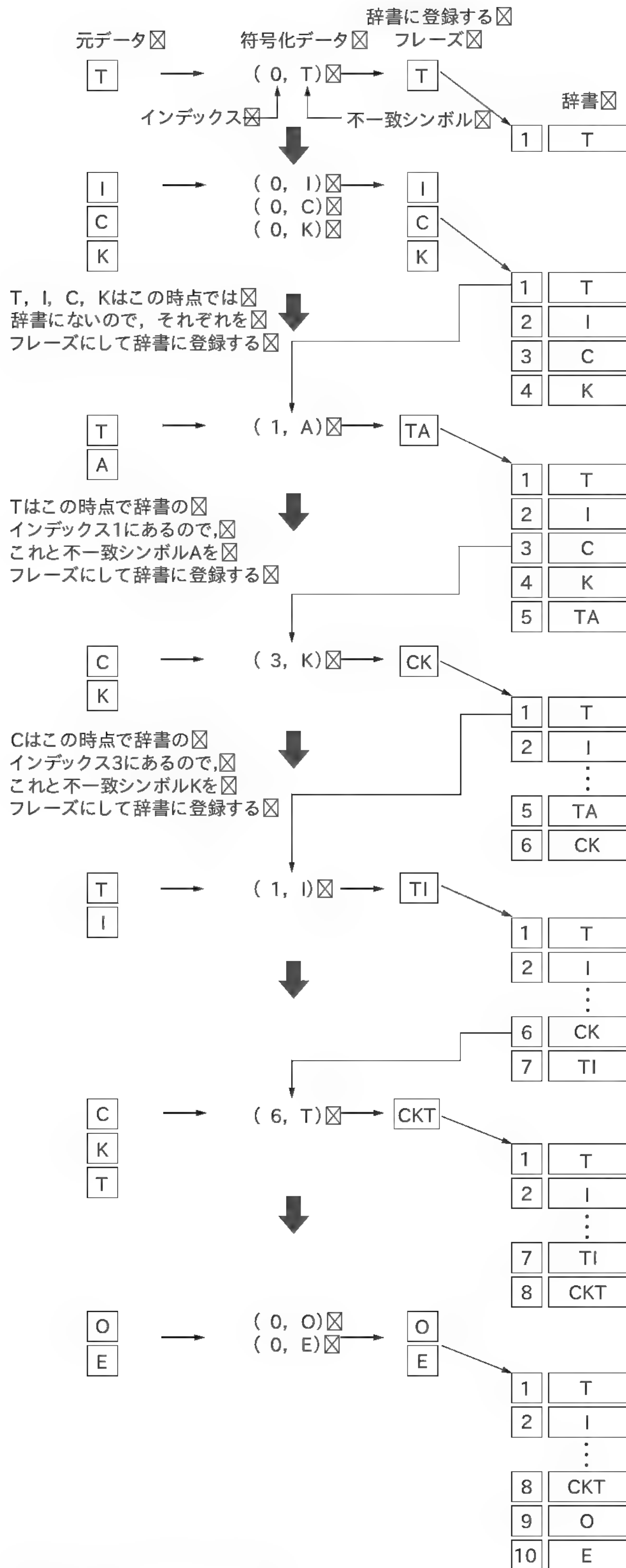


図3　動的辞書圧縮の符号化

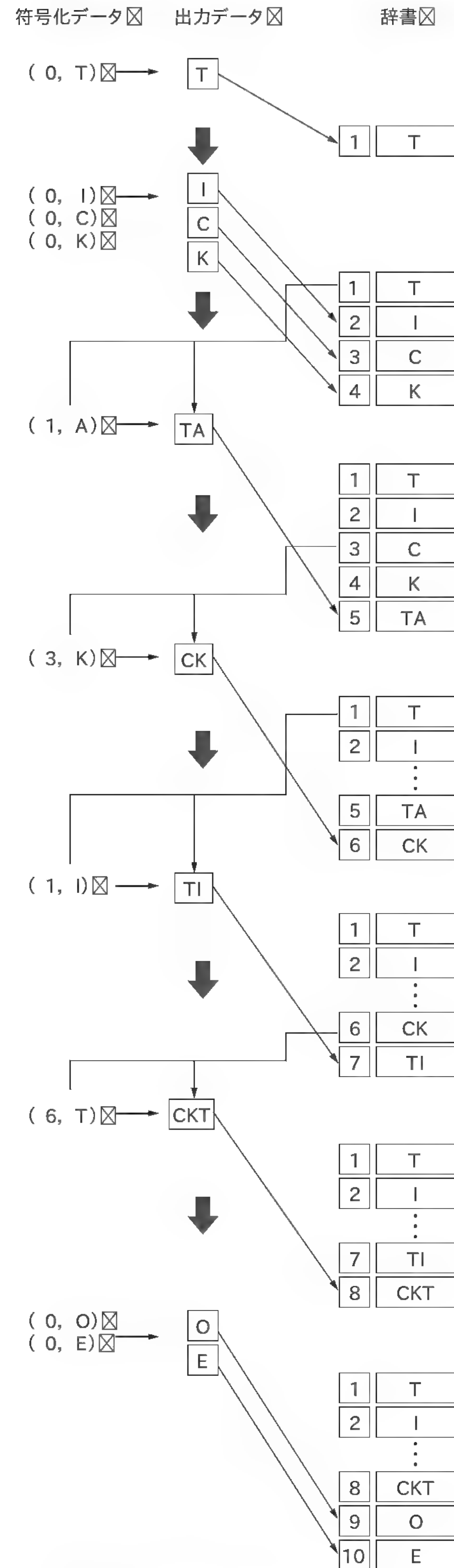


図4　動的辞書圧縮の復号化

リスト 5　動的辞書圧縮の符号化メイン処理

```
//不一致シンボルの表現ビット数
#define LZ_DD_NEXTBYTE_BITS_SIZE 8

//動的辞書圧縮のクラス
class LZ_DD_Compress {
public:
    //圧縮を行う.
    // 成功すれば圧縮後のサイズを返す，失敗すれば0を返す
    //iRBB=読み取りオブジェクト，iWBB=書き込みオブジェクト，
    // iStartPos=読み取り開始位置(0～)，
    // iProcSize=圧縮対象のサイズ
    uint_32_t compress(ReadByteBase& iRBB,WriteByteBase& iWBB,
                    uint_32_t iStartPos,int_32_t iProcSize) {
        //サイズが0以下なら戻る
        if(iProcSize <= 0)
            return 0;
        //書き込み済みサイズをリセット
        iWBB.reset_wrote_size();
        //圧縮対象サイズを書き込む
        if(!iWBB.write_4byte(iProcSize))
            return 0;
        //bit単位書き込みオブジェクトを用意する
        WriteBitClass aWBits(iWBB);
        //符号化ヘルパ・オブジェクトを発生する
        LZ_DD_CompressHelper aHelper;
        //符号化すべき残りバイト数
        int_32_t aRemainBytes = iProcSize;
        //処理すべき読み取りデータがある限り，符号化を繰り返す
        while(aRemainBytes > 0){
            uint_8_t aSymbol;
            //1バイトを読み取る
            if(!iRBB.read_1byte(aSymbol))
                return 0;
            --aRemainBytes;
            //符号化を試みる
            if(aHelper.push(aSymbol,aRemainBytes == 0)){
                unsigned int aPopBits = aHelper.popBits();
                unsigned int aPopIndex = aHelper.popIndex();
                uint_8_t aPopSymbol = aHelper.popSymbol();
                //辞書インデックスを書き込む
                if(aPopBits > 0){
                    if(!aWBits.write_bits(aPopBits,aPopIndex))
                        return 0;
                }
                //不一致シンボルを書き込む
                if(!aWBits.write_bits(
                        LZ_DD_NEXTBYTE_BITS_SIZE,aPopSymbol))
                    return 0;
            }
        }
        //bit単位書き込みオブジェクトをフラッシュする
        if(!aWBits.close_buffer())
            return 0;

        return iWBB.get_wrote_size();
    }
```

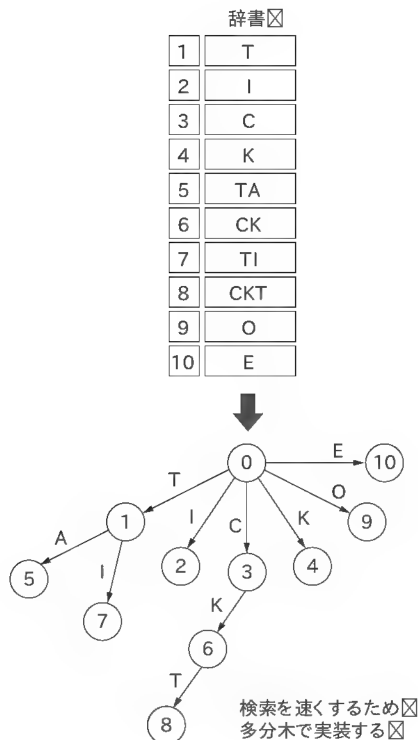


図5
符号化で利用する
多分木

するたびに辞書が確実に1フレーズずつ成長するので，インデックス番号の表現ビット数を変更することはさほど難しくありません注4.

ところで辞書を育てるのが動的辞書圧縮の特徴ですが，反面，どこまで育ててよいかが問題です．辞書を記録するメモリや記憶媒体は有限ですから，いずれこれ以上は記録できない上限に達するはずです．また，あまりにも巨大な辞書は検索も登録も時間がかかりますし，符号化されたものはビット数が多いので圧縮率を悪くする可能性があります．そのため辞書がある程度以上大きくなったなら，

(1) 辞書の登録や更新を停止し，ここから以降は停止した時点の内容，つまり静的辞書として取り扱う
(2) 出現頻度の変化を考え，いったんこの時点で辞書の全内容を破棄して，ゼロから再出発する

という二つの考えがあるようです．ちなみに筆者が今回作成した実装例では，辞書が大きくなったときの対策をしていないので，流用するときは，この点に注意を払ってください．

動的辞書圧縮の実装例

符号化処理のメインはリスト5のようになります．ここでも最初に圧縮前のサイズを書き込みますが，やはり復元するときの目安として必要だからです．

符号化するとき，読み取ったフレーズがすでに辞書に登録されているかどうかを調べる必要がありますが，それは一筋縄ではいきません．なぜなら，登録されているフレーズは可変長サイズであるので，1バイトずつ読み取ったものを辞書に登録されたフレーズと一致しているか力まかせに比較していく下手なプログラムになりかねないからです．ここではヘルパ・オブジェ

注4：辞書にあるフレーズ数を格納できる2の累乗．2を底とする対数で求められる．たとえば最大インデックス番号が8から15までなら，表現可能なインデックス番号の総数は8(2の3乗)より大きく，16(2の4乗)より小さいか等しいので，表現ビット数は4ビット必要となる．

リスト 6　多分木ノード・クラスと符号化ヘルパ

```
//符号化で使う多分木ノード・クラス
class LZ_DD_TreeNode {
    typedef std::map<uint_8_t,LZ_DD_TreeNode*> TreeNodeMap;

    unsigned int mIndex; //インデックス番号
    TreeNodeMap mTreeNodeMap; //別ノードへのポイント情報
public:
    LZ_DD_TreeNode(unsigned int iIndex = 0) {
        mIndex = iIndex;
    }
    ~LZ_DD_TreeNode() {
        TreeNodeMap::iterator aItr;
        for(aItr = mTreeNodeMap.begin();
                aItr != mTreeNodeMap.end(); aItr++){
            LZ_DD_TreeNode* aTreeNode = aItr->second;
            delete aTreeNode;
        }
    }
    //ノードに割り当てられたインデックス番号を返す
    unsigned int getIndex() const {
        return mIndex;
    }
    //ノードにシンボルを追加登録する
    void appendSymbol(uint_8_t iSymbol,unsigned int iIndex) {
        mTreeNodeMap.insert(std::make_pair(iSymbol,
                            new LZ_DD_TreeNode(iIndex)));
    }
    //指定シンボルと関連づけられているノードを返す
    //ない場合はNULLを返す
    LZ_DD_TreeNode* getNode(uint_8_t iSymbol) {
        TreeNodeMap::iterator aItr = mTreeNodeMap.find(
                                                iSymbol);
        return (aItr == mTreeNodeMap.end()) ?
                            NULL : aItr->second;
    }
};

//表現するのに必要なビット数を返す
inline unsigned int needs_bits(uint_32_t iNum)
{
    uint_32_t aMask = 1;
    unsigned int aResult;
    for(aResult = 0; aResult < 32; aResult++){
        if(aMask > iNum)
            return aResult;
        aMask <<= 1;
    }
    return aResult;
}

//符号化を助けるクラス
class LZ_DD_CompressHelper {
    unsigned int mDictSize;    //辞書サイズ
    LZ_DD_TreeNode* mRootNode; //多分木のルート・ノード
    LZ_DD_TreeNode* mNodePtr;  //多分木をたどるポインタ
    uint_8_t mNextByte;        //不一致シンボル
    unsigned int mOrigDS;      //pushを呼び出した直後の辞書サイズ
public:
    LZ_DD_CompressHelper() {
        mDictSize = 0;
        mRootNode = new LZ_DD_TreeNode();
        mNodePtr = mRootNode;
    }
    ~LZ_DD_CompressHelper() {
        delete mRootNode;
    }
    //シンボルをヘルパに教える
    //このメンバ関数がtrueを返す限り，符号化は終わっていない
    //trueを返したなら，popで始まるメンバ関数で符号を
    // 取り出すこと
    //iSymbol=シンボル,iLast=trueならこのシンボルが最後のもの
    //戻り値=true:まだ多分木をたどれる，
    //       false:これ以上は多分木をたどれない
    bool push(uint_8_t iSymbol,bool iLast) {
        mOrigDS = mDictSize;
        //最後のシンボルであるなら，
        // ここで多分木の探索は打ち切る
        if(iLast){
            mNextByte = iSymbol;
            return true;
        }
        //多分木に登録されているシンボルであるかを調べる
        LZ_DD_TreeNode* aNodePtr = mNodePtr->getNode(iSymbol);
        //されていないなら，ここで打ち切る
        if(aNodePtr == NULL){
            mNextByte = iSymbol;
            //多分木に新しいノードを追加
            mNodePtr->appendSymbol(iSymbol,++mDictSize);

            return true;
        }
        //登録されているなら，まだ打ち切らない
        mNodePtr = aNodePtr;

        return false;
    }

    //popで始まるメンバ関数はpushがtrueを返したときのみ
    // 呼び出すこと
    //順番はpopBits→popIndex→popSymbol

    //インデックスの表現可能ビット数(0～)を返す
    unsigned int popBits() const {
        return needs_bits(mOrigDS);
    }
    //インデックスを返す
    unsigned int popIndex() {
        unsigned int aIndex = mNodePtr->getIndex();
        //多分木ポインタをルートに戻す
        mNodePtr = mRootNode;

        return aIndex;
    }
    //不一致シンボルを返す
    uint_8_t popSymbol() const {
        return mNextByte;
    }
};
```

クト(LZ_DD_CompressHelper)を導入し，1バイトずつ読み取ったものをヘルパ・オブジェクトに教えていくという，一見まどろっこしい手法を取ります．しかし，この手法のほうが結果的に簡素なプログラムになります．ヘルパ・オブジェクトはリスト 6のように実装します．

ヘルパ・オブジェクトの仕事は，pushメンバ関数で教えられた1バイトずつの元データが辞書に登録されているかどうかを調べることです．辞書の検索は高速化のために多分木を構築するようにしています(図5)．こうすることで任意サイズのフレーズを検索しやすくすると同時に新たなフレーズの登録も手早くすむようになります．

一方，復号化はリスト 7のようになります．ここでもヘルパ・オブジェクトを導入することで処理を簡素にするようにくふうしています．

復号化は符号化と違い，辞書を多分木にする必要はありませんが，手早く辞書を構築するくふうは必要でしょう．そのための単語クラス(LZ_DD_DictToken)とヘルパ・オブジェクト(LZ_DD_ExpandHelper)をリスト 8のように実装します．

以上で動的辞書圧縮の実装は完成です．

みやさか・でんと　miyadent@anet.ne.jp

リスト 7　動的辞書圧縮の復号化メイン処理

```
class LZ_DD_Compress {
public:
    ...(途中省略)...
    //展開を行う，成功すれば展開後のサイズを返す，
    // 失敗すれば 0 を返す
    //iRBB=読み取りオブジェクト，iWBB=書き込みオブジェクト，
    // iStartPos=読み取り開始位置(0～)
    uint_32_t uncompress(ReadByteBase& iRBB,
                    WriteByteBase& iWBB,uint_32_t iStartPos){
        //書き込み済みサイズをリセット
        iWBB.reset_wrote_size();
        //読み取り開始位置を iStartPos に移動
        if(!iRBB.rewind_ptr(iStartPos))
            return 0;
        //最初の 4 バイトを読む （復元サイズ）
        uint_32_t aOrigSize;
        if(!iRBB.read_4byte(aOrigSize))
            return 0;
        int_32_t aRemainBytes = static_cast<int_32_t>(
                    aOrigSize); //復元すべき残りバイト数
        //復号化ヘルパを発生する
        LZ_DD_ExpandHelper aHelper;
        //bit 単位読み取りオブジェクトを用意する
        ReadBitClass aRBits(iRBB);
        //復元すべき読み取りデータがある限り，復元を繰り返す
        while(aRemainBytes > 0){
            //インデックスを読み取る
            unsigned int aPopBits = aHelper.popBits();
            uint_32_t aIndex;
            if(aPopBits != 0){
                if(!aRBits.read_bits(aPopBits,aIndex))
                    return 0;
            }
            //不一致シンボルを読み取る
            uint_32_t aNextByte;
            if(!aRBits.read_bits(LZ_DD_NEXTBYTE_BITS_SIZE,
                                aNextByte))
                return 0;
            //辞書から取り出した単語を書き込む
            LZ_DD_DictToken* aToken = aHelper.push(aIndex,
                        static_cast<uint_8_t>(aNextByte));
            if(aToken != NULL){
                unsigned int aLength = aToken->getLength();
                for(unsigned int aI = 0; aI < aLength; aI++){
                    if(!iWBB.write_1byte(aToken->
                                    getSymbol(aI)))
                        return 0;
                }
                aRemainBytes -= aLength;
            }
            //不一致シンボルを書き込む
            if(!iWBB.write_1byte(static_cast<uint_8_t>(
                                    aNextByte)))
                return 0;
            --aRemainBytes;
        }

        return iWBB.get_wrote_size();
    }
};
```

リスト 8　単語クラスと復号化ヘルパ

```
//復号化で使う単語クラス
class LZ_DD_DictToken {
    uint_8_t* mToken; //単語を格納する
    unsigned int mLength; //単語の長さ

    LZ_DD_DictToken(); //(empty)
public:
    LZ_DD_DictToken(uint_8_t iSymbol){
        mToken = new uint_8_t[1];
        mToken[0] = iSymbol;
        mLength = 1;
    }
    LZ_DD_DictToken(const LZ_DD_DictToken* iDictToken,
                    uint_8_t iSymbol){
        mLength = iDictToken->mLength + 1;
        mToken = new uint_8_t[mLength];
        std::memcpy(mToken,iDictToken->mToken,
                    iDictToken->mLength);
        mToken[iDictToken->mLength] = iSymbol;
    }
    ~LZ_DD_DictToken(){
        delete[] mToken;
    }
    unsigned int getLength() const {
        return mLength;
    }
    uint_8_t getSymbol(int iIndex) const {
        return mToken[iIndex];
    }
};

//復号化を助けるクラス
class LZ_DD_ExpandHelper {
    typedef std::vector<LZ_DD_DictToken*> DictType;
    DictType mDict; //辞書
public:
    ~LZ_DD_ExpandHelper() {
        for(DictType::iterator aItr = mDict.begin();
                aItr != mDict.end(); aItr++){
            LZ_DD_DictToken* aToken = *aItr;
            delete aToken;
        }
    }
    //インデックスの表現可能ビット数(0～)を返す
    unsigned int popBits() const {
        return needs_bits(mDict.size());
    }
    //符号を復元するための手がかりを返す
    //iIndex=インデックス,iNextByte=不一致シンボル
    //戻り値=辞書に登録した単語,見つからないなら NULL
    LZ_DD_DictToken* push(unsigned int iIndex,
                        uint_8_t iNextByte) {
        LZ_DD_DictToken* aResult;
        LZ_DD_DictToken* aNewToken;
        if(iIndex == 0){
            aResult = NULL;
            aNewToken = new LZ_DD_DictToken(iNextByte);
        }else{
            aResult = mDict[iIndex - 1];
            aNewToken = new LZ_DD_DictToken(aResult,iNextByte);
        }
        mDict.push_back(aNewToken);

        return aResult;
    }
};
```



はじめに

CRC(Cyclic Redundancy Check)は有名な誤り検出符号です．誤り検出・訂正の分野では，もはや古典ともいえる技術ですが，手軽で実用度も高いため，ネットワーク通信やファイルのデータ・チェックをはじめとして，今でも至るところで使われています．

CRC計算回路は，はるか昔から多くの教科書や書籍で紹介されていますが，本稿では回路をアプリケーションに応じてアレンジすることをテーマにします．そのためには，既存の回路をただまねるのではなく，「いったいどんな意味の処理を行っているのか」を掴(つか)むことがやはり必要になります．ここでは実際の処理内容について，やさしく説明します．

1 CRC計算の意味を直感的に理解しよう

● まずは常識のおさらい

CRCは，ネットワークなどでデータを転送するときに，受け手が「正しいデータが来たかどうか」を判定できるようにするためのメカニズム(誤り検出符号注1)です．送り手のほうでは，誤りが検出できるような形にデータを加工してから送信を行います．

その根本にある発想は，受け手がnビットのデータを受けるとして，その値の範囲0～2^n-1を，正しいデータの領域とまちがったデータの領域の二つに分けておくことです(図1)．

送り手は，送りたいデータ(情報語)に付加ビット(通常パリティと呼ばれる)を付けるなどの方法でビット数を増やし，そうした領域の区別を作ったうえで，正しいデータ領域(符号語という)の値のみを送るようにします．受け手では，受け取ったデータ(受信語)がどちらの領域にあるかを判定します．

この領域の具体的な分け方は，エラー検出能力や実装コストなど，いろいろな要素を考え合わせて決めます．

● 基本原理は「割り切れる数だけを送る」こと

実際のCRCとは少し違いますが，まずはわかりやすい整数を使って，CRC計算処理の意味合いを直感的に捉えてみることにしましょう．

CRCの場合，送り手はある与えられた数Gで割り切れるデータのみを送り，受け手ではGで割り切れるか否かでエラーの有無を判定します．たとえば，Gを100と決めたならば，送り手は0, 100, 200, 300, …のみを送ります．受け手では，これらの数が来ればOK，99や101といった数が来ればエラーと判定します．もしデータ転送の過程でエラーが発生すれば，たいていはGで割り切れない数にデータが変化してしまうので，それでエラーを検出できるというしくみです注2．すなわち，図1の

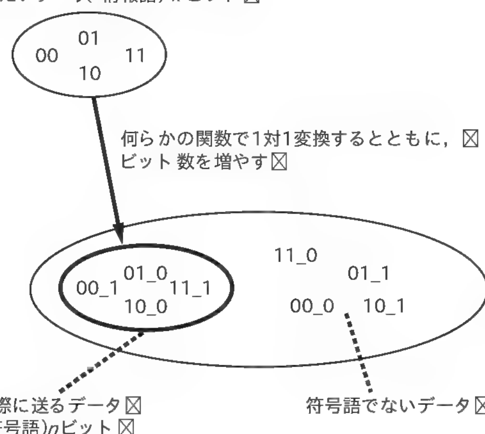


図1 エラー検出の原理
送り手では，もとのデータを何らかの方法でビット拡張し，符号語のデータとそうでないデータという二つの領域を作る．受け手では，符号語でないデータがきたらエラーと判断する

注1：誤りがあるかないかだけを判定できるのではなく，誤りがあればそれを修正できるようなメカニズムを誤り訂正符号と呼ぶ．CRCでも，発生するエラーのタイプをかなり限定すれば誤り訂正は可能だが，通常は誤り検出のみの用途に使う．

注2：つごうのよい(？)形のエラーが発生して，もとの値とは違うが100で割り切れる値に化けてしまう(つまり，ほかの符号語に化けてしまう)と，エラー検出できない．これはCRCに限らずどのような誤り訂正・検出符号であっても起こる本質的な問題である．そこで，起こりうるエラーのタイプをあらかじめ想定しておき，そのもとで問題が発生しないような符号を設計・選択して使う．

データ領域を，Gで割り切れるか否かによって二つに分けます．

● 送信データを得る基本手順

さて，本当に送りたいデータ，つまり情報語がG(ここでは100)でつねに割り切れるとは限りません．99や101を送りたい場合もあります．

そこで，情報語Mから，実際に送信するためのデータ，つまり符号語を新しく作ります．実際のCRCによく似た作成法を図2に示します．整数の上で説明しようとすると，多少むだに思える部分や実際のCRCと異なる部分が出てきてしまいますが，まずは雰囲気を掴(つか)んでください．

(1) 情報語Mを何倍か(G以上)してやる．すると，もとのデータが0，1，2，3，…と連続であっても，たとえば0，6，12，18，…と飛び飛びの値になる．これによって，図1での「正しいデータ」と「正しくないデータ」の二つの領域ができる．もちろんデータのビット数は増える

(2) (1)の結果M'をGで割った余りRを求める．

(3) M'からRを引く．その結果は明らかにGで割り切れる．たとえば，M'=18，G=5のときR=18 mod 5=3だが，$M'-R$=15でGで割り切れる

(4) (3)の値を送信する．送信データはもとのデータMと1対1に対応する

(5) 受け手では，送られてきたデータがGで割り切れるかを調べる

● ここまでのやり方で検討が足りない点

以上は整数のうえで考えた話ですが，実際のエラー検出に使おうとするならば，次の点を再検討しなければなりません．

(1) エラーの検出能力

データは2進値として送ることになりますが，たとえば，あるビットのエラー(反転)は検出しやすいが，あるビットはしにくい，というようでは困ります．そのため，符号語のビット・アサインの方法などをきちんと考える必要があります．また，どのようなタイプのエラーが検出できるかがきちんと数学的に分析できていないと，目的とするアプリケーションに使ったときに効果があるか，有効かどうかの判断材料がありません．

(2) 受信側で情報語Mを取り出せるか

受け取ったデータのエラー検出ができるだけでは不十分で，もとのデータMをなるべく簡単に取り出せる必要があります．

(3) 演算処理のコスト

送りたいデータは，数十ビットや数百ビットというように多ビットになるのが普通です．こうした多ビット長演算は，プログラムや回路で作ると高コストになる場合があります．たとえば，整数の割り算はかなりコストのかかる演算です．

以上の点を整数上で検討あるいは解決するのが厄介なのは，容易に想像がつくと思います．ところが，次で説明するような「誤り検出・訂正に独特な数体系」を導入すると，驚くほどすっきりと解決されます．

2 実際のCRCの計算方法

それでは，実際のCRC計算の方法を説明していきます．1節で整数を使ってあらすじを説明しましたが，それと異なる部分に絞って話を進めます．

● データを整数ではなく多項式とみなす

1節ではデータを整数値とみなしていましたが，実際にはデータを「ガロア体$GF(2)$上の多項式[注3]」とみなします(「モジュロ2の演算」と呼ばれることもある)．

ガロア体などの言葉を聞きなれていないと非常に難しいことのように思えるかもしれませんが，とても簡単です．図3に例を示します．nビットのデータを$n-1$次の多項式に対応づけ，i次の係数がデータのビットiの値(0か1)になるようにします．1節におけるM，M'，G，Rなどのデータはすべて，整数ではなく多項式として扱います．

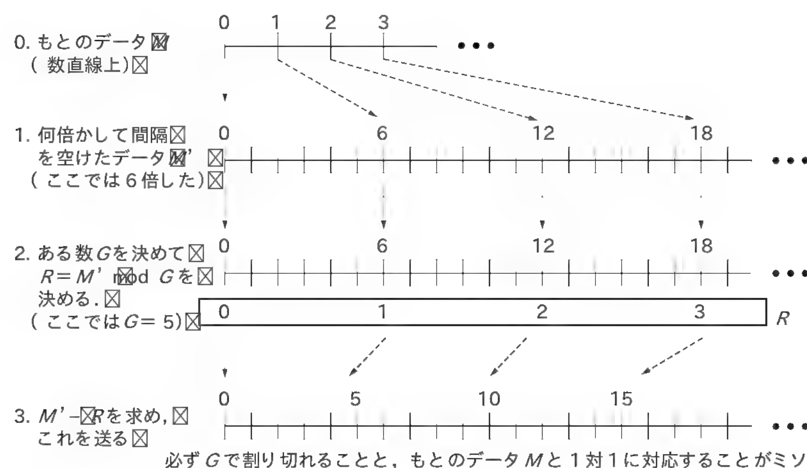


図2　CRC計算とかなり似た整数上の符号語生成法(実際のCRCは整数演算ではないので詳細は異なる)

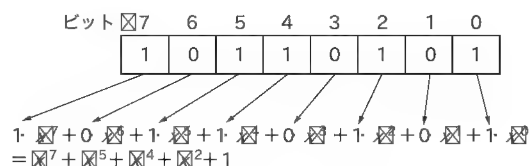


図3　データを$GF(2)$上の多項式と見なす
nビットのデータは(最高)$n-1$次の多項式で表現される．xの次数がビット位置に対応し，xの係数はビット値(0か1)である

注3：xの係数がガロア体$GF(2)$の要素，つまり{0, 1}であることから「$GF(2)$上の多項式」と呼ばれる．なお，ガロア体は$GF(2)$だけでなくいろいろな種類があり，一般には複数ビット数の値をとる．誤り検出・訂正符号でも，$GF(2)$のみならず，その拡大体と呼ばれるものが良く使われる．詳しくは参考文献(1)などを参照のこと．

わざわざこのようなことをする理由は，1節の最後に示した数々の問題をすべてきれいに解決できるからです．すなわち，

- 多項式表現とビット・レベル・データが一対一に対応しているとともに，ガロア体上の多項式の演算に関しては多くの数学的研究があるため，符号がビット・レベルでどのような性質を持つかを数学的に分析しやすい(詳しくは参考文献3)などの符号理論の教科書を参照)
- 受信時にもとのデータの分離が容易(詳細は後述)
- 演算処理を低コストで実装できる(詳細は後述)

というメリットが出てくるからです．

● 多項式どうしの割り算の方法

このようにデータの表現の方法は違っても，1節で示したような割り算を行うこと自体は同じです．では，GF(2)上の多項式の割り算とは一体どのようなものか，見ていきましょう．

図4に，GF(2)上の多項式どうしの割り算の例を示します．普通の整数上の多項式で割り算を行う場合と同じように，筆算していくだけです．

(1) 被除数の最高次の係数(要するにMSB)の値を見て，1ならば除数の桁を合わせて引く

(2) 桁を一つ右に(LSB側へ)ずらし，(1)に戻る

ただし，次の二点は整数とまったく異なります．

```
多項式x^5+x+1("データ" 100011")を
多項式x^3+x+1("データ" 1011")で割った例

STEP1: 商を1ビットぶん求める
                 1
        ---------------
1 0 1 1 ) [1] 0 0 0 1 1
           └ MSBがそのまま商になる

STEP2:「除数×求めた商」を引く
                 1
        ---------------
1 0 1 1 ) [1 0 0 0] 1 1
          [1 0 1 1]
        ---------------
          [0 0 1 1] 1
           └ 桁ごとにXOR．桁借りはない

STEP3以降: 桁をずらして繰り返し
             1 0 1          商
        ---------------
1 0 1 1 )  1 0 0 0 1 1
           1 0 1 1
        ---------------
            [0] 1 1 1
商のビット1 ─┘ 0 0 0 0
        ---------------
               [1] 1 1 1
商のビット0 ────┘ 1 0 1 1
        ---------------
                   1 0 0  余り(3ビット)
```

商は x^2+1，余りは x^2

検算: 除数×商+余り

$=(x^3+x+1)(x^2+1)+x^2$

$=x^5+x^3+x^2+x^3+x+1+x^2$

$=x^5+x+1=$ 被除数

図4　GF(2)上の多項式の割り算の例
整数上の多項式の場合と同じように筆算すればよい．ただし，係数部の演算の方法は整数とは異なる．また，多項式の演算なので，上位桁からのボローもない

- 桁ごとに独立に係数の減算を行い，上位桁からのボロー(桁借り)はない
- 係数の減算は整数の引き算ではなく，XORで行う．すなわち，0−0=0，0−1=1，1−0=1，1=1=0となる．なお，係数の加算もXORで，減算と加算は等価

● CRC計算の全体像

図5に，CRC計算の実際の手順を示します．1節の例における*M*，*G*，*R*などは，すべて多項式となります．以降，これらを*M*(*x*)，*G*(*x*)，*R*(*x*)と書きます．*M*(*x*)は情報語，*G*(*x*)は生成多項式，*R*(*x*)はパリティです．整数の場合(図1)と異なるのは次の点です．

- 整数ではステップ1でもとのデータを何倍かしたが，これはデータをMSB側へシフトして多項式の次数を上げることで行う．シフトのビット数は，パリティ長，すなわち*G*(*x*)の次数と同じ
- 整数のステップ3で*M*'−*R*を求めたが，これは*M*(*x*)−*R*(*x*)を求めることで行う．この値は，もとの情報語*M*(*x*)にパリティ*R*(*x*)を連接することによっても求められる．整数の場合と異なり，符号語において情報語とパリティがビット・レベルで分離しているので，受け手で情報語を切り出しやすいという大きなメリットがある

● CRC計算のいろいろな実装バリエーション

以上でCRC演算の中心的な部分の説明は終わりですが，実用でCRC演算を用いるときには，次のような点で演算の方法にバリエーションがあるので，必ずターゲット・アプリケーションの仕様書を調べてください．

(1) 生成多項式

これによってパリティ長が変わります．おもなものについては，表1を参照してください．

(2) 入力データのビット順

これまでは入力データ*M*(*x*)のMSBが多項式の高次数側に

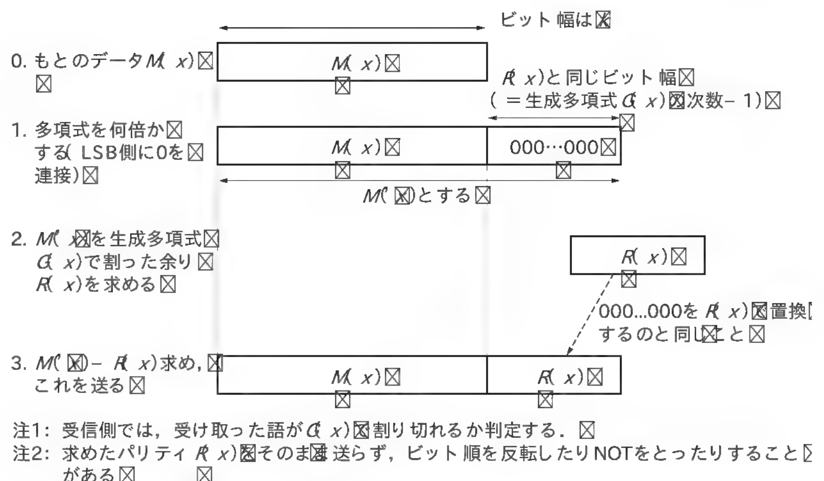


図5　CRC計算の実際
各ステップは図1の整数の場合とおおむね対応しているが，データ操作の具体的なやり方には違いがある

表1　実用に使われているおもなCRC

名　称	生成多項式	パリティ長	おもな用途
CCITT-32 /CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$	32	Ethernet, FDDI, PKZIP
CRC-16	$x^{16}+x^{15}+x^{2}+1$	16	USB
CRC-CCITT	$x^{16}+x^{12}+x^{5}+1$	16	HLDC/SLDC
CRC-12	$x^{12}+x^{11}+x^{3}+x^{2}+x+1$	12	
CRC-10	$x^{10}+x^{9}+x^{5}+x^{4}+x+1$	10	
CRC-8	$x^{8}+x^{2}+x+1$	8	
CRC-5	$x^{5}+x^{2}+1$	5	USB

あたるとして説明してきましたが，逆になる場合(LSBが高次数)があります．

(3) CRC計算の初期値

CRC計算に用いるレジスタ(次節以降で説明)を計算開始時に初期化する値です．そのビット数はパリティ長と同じで，多くの場合，値はオール0かオール1です．

(4) パリティのビット順

計算したパリティ $R(x)$ をそのまま $M(x)$ に連接するのではなく，MSBとLSBを反転してから連接する場合があります．

(5) パリティへのXOR値

パリティ $R(x)$，またはそのビット順を反転したものに対して，ある固定値をXORしてから $M(x)$ に連接する方法です．

これらの方法が使われている場合，本来のCRC計算方法そのままでは処理できないことがあるので，対応については4節の最後で説明します．

なお，上記(3)についてはおもしろいので簡単に説明します．扱うデータが可変長で，ビット落ちがあるような場合を考えます．ここで，CRCのような除算では，データの先頭部の0はパリティ値に影響を与えません．データが先頭の0が1000個で最後が01のデータでも，0が100個で最後が01のデータでも，パリティ値は同じなのです．このようなデータでは，先頭の0がいくつか欠落しても受け手でそれを検出できません．計算前にパリティの初期値を0以外の値にしておくことによって，これを防止できます．

3 もっとも基本的なCRC回路

● 筆算一段分を組み合わせ回路で作るのが基本

回路をいろいろアレンジして作るうえでの基礎となるのが，図4で示した筆算の一段分を作る方法です．図6に，その方法を示します．回路と筆算の式とは1対1に対応しているので，よく見比べれば理解できます．回路へは，$M'(x)$(情報語に0を連接したもの)が，MSBから順に1ビットずつやってくるものとします．以下，回路構成の説明です．

- 余りを計算・格納するためのレジスタを持つ．図6の例では，生成多項式は3次(4ビット)なので余りは3ビットである．そこでレジスタも3ビット持ち，そこに筆算における上位3

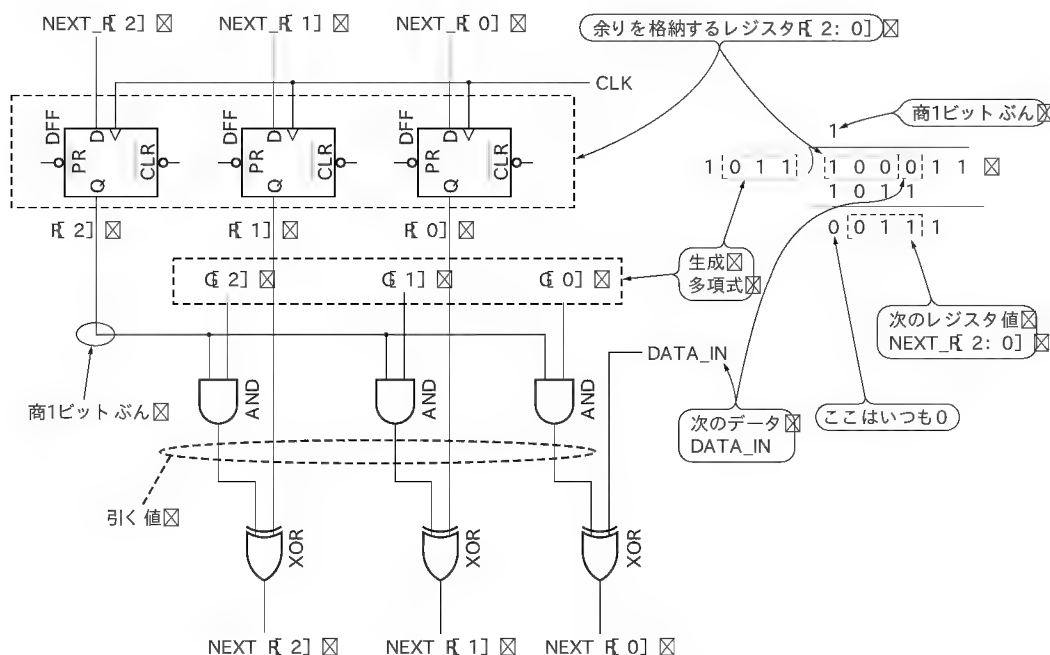


図6　多項式除算の筆算1段ぶんの回路を作る(生成多項式は任意の3次式，パリティは3ビット)

筆算と回路は1対1に対応している．被減数の先頭部分をレジスタに持つ．商の値によって引く数を決定するのがANDゲート群であり，引き算を行うのがXORゲートである．引いた結果を1ビット左にずらして再度レジスタへストアする．この回路は生成多項式の切り替えが可能である

ビットぶんを格納する

- 2節で説明したとおり，レジスタの最上位ビットが商になる．その商の値と生成多項式の値から，引く値を求めるのがANDゲート群．ANDの出力は，商が1なら生成多項式の値，商が0なら0となる
- 引き算を行うのがXORゲート群．生成多項式の最上位ビットについては，引き算をしたところで結果（つねに0）は捨ててしまうので，回路を用意しない．3ビットぶんだけ引き算をする
- 引き算した結果を，1ビット左シフトしてレジスタの次の値とする

● 回路の動かし始めと動かし終わりに注意

前節の回路（図6）を実際に使うにあたっては，その動かし方にいくつか注意すべき点があります．図7に回路の動かし方を示します．

まず，送信側については次のようにします．

（1）CRC計算を始める前に，レジスタをすべて0クリアする

（2）次に情報語$M(x)$（送りたいデータ）を，MSBから順に1クロックに1ビットずつ投入する．普通は，これと並行して情報語を受け手へシリアル送信する

（3）情報語をすべて投入したら，続けて0をパリティ$R(x)$のビット数ぶんだけ投入する

（4）すべての0を投入し終わった直後のレジスタ値がCRCとなる．これを受け手へ送信する．CRCの値はパラレルに取り出してもかまわないし，レジスタ値を左シフトしながらシリアルに取り出してもかまわない．シリアルに取り出すときは，生成多項式の値をレジスタから減算しないよう，図6のAND出力を強制的に0にする必要がある

次に，受け手については，回路のデータ・パスは送信側と同じ（図6）ですが，その制御が若干異なります．

（1）データの受信を始める前に，レジスタを0クリアしておく

（2）受信したデータ（パリティを含む）をMSBから順に投入する

（3）パリティの最後のビットを投入し終わった直後にレジスタ値が0になっていればエラーはない．0でなければエラーあり

● 基本回路における問題点

さて，図6の基本回路を図7に従って実際に使おうとすると，次のような扱いにくい点があることがわかります．

- 送信側において，情報語を投入してから0を投入しなければならない．このため，情報語$M(x)$の最後のビットを投入し終わってからパリティ$R(x)$の値が得られるまでに，何クロックか時間が空いてしまう．ところが実用では，情報語に続けてすぐパリティを送信したい場合が多く，情報語の送信を少し遅らせるなどのくふうが必要になってしまう
- データの送受信を1ビットずつシリアルで行うことになる．しかし，実用ではデータが数ビット単位で一度に来る場合も多く，パラレル-シリアル変換やシリアル-パラレル変換が必要になってしまう

これらの修正については，次節で紹介します．

4 実用的なCRC回路

● 高速CRC…もっともよく使われる実装方法

前節の最後で述べた問題点の一つ，「送信側では，情報語に続いて0を投入しなければならない」を解決し，0を投入しなくても済む回路を考えてみることにします（図8）．

この回路の構成にあたっては，まず図6の回路がどのような演算を行うかを見直してみることが役に立ちます．あるデータ$N(x)$を図6の回路に投入し終わった時点で，回路のレジスタに入っている値は$N(x) \bmod G(x)$です．このことから，図6の回路は，1クロックごとに図8（a）のとおりにレジスタ値を変化させることがわかります．

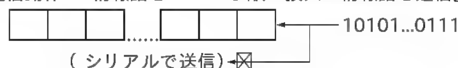
さて，問題を解決するには，情報語$M(x)$を投入し終わった時点で，レジスタに$M(x) \cdot x^r \bmod G(x)$（$r$はパリティのビット数）が入っているような回路を作ればよいわけです．すなわち，$N(x)$をそれまでに入力された部分データとして，図8（b）のようにレジスタ値を変化させればよい，ということです．

ここまでくれば，あとは算数の知識を少し使えば欲しい回路が作れます．図8（b）における次レジスタ値は，mod演算の線形性［$A(x)+B(x) \bmod G(x) = A(x) \bmod G(x) + B(x) \bmod G(x)$］を使うと，二つのmod演算に分解されます．これらは，図8（c）のとおり，並列に並べた二つの基本CRC回路

送信動作1．レジスタをゼロ・クリア☒



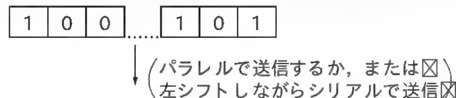
送信動作2．情報語をMSBから順に投入＆情報語を送信☒



送信動作3．パリティのビット数ぶんの0を投入☒



送信動作4．すべての0を投入しおわった時点の値がCRC



受信動作1．レジスタを0クリア☒



受信動作2．受信データ（含パリティ）をMSBから順に投入☒



受信動作3．すべて投入しおわった時点の値がALL0ならOK



図7　もっとも基本的な送信/受信回路の動作（実際にはいろいろなアレンジが必要）
送信側では，送りたいデータ$M(x)$（情報語）に引き続いて0をパリティのビット数ぶん投入しなければいけないことに注意

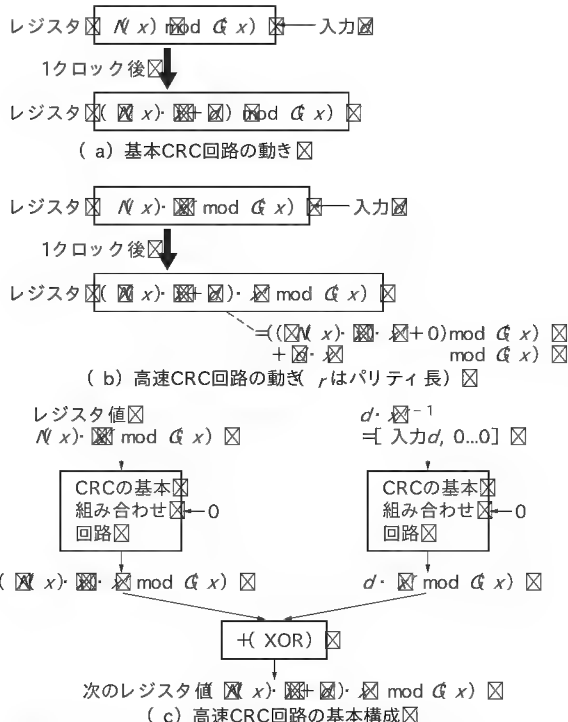


図8　最後に0を投入しなくて済む高速CRCの基本アイデア

それまでに入力された多項式 $N(x)$ に対して，レジスタにつねに $N(x) \cdot x^r \bmod G(x)$ が入っているようにする．これは，mod演算の線形性を利用すると，基本CRC回路（図6）を二つ並べることによって実現できる

（図6）で演算させることができます．図9に，生成多項式を3次とした場合の回路の例を示します．

なお，受信側でも同じ回路が使えます．エラー検出は，3節の基本回路と同じく，レジスタが0になったかどうかで行います．

● 生成多項式を固定すれば回路は簡単に

図6や図9の回路は，どのような生成多項式にも使え，生成多項式の切り替えも可能です．少しくふうすれば，さまざまな次数の生成多項式を使えるように改造することもできるので，考えてみてください．

しかし，生成多項式を一種類しか使わないのであれば，その値を固定値として図6と図9の回路に代入したうえでロジックを整理すると，図10のようにいたってシンプルな回路になります．これらは，符号理論の教科書などでよく紹介されている回路そのものです．

なお，XORを取る位置は「次レジスタ値のうち，生成多項式の係数が1のビット位置」です．図10では生成多項式は $x^3+x+1=x^3+x^1+x^0$ なので，x^1 と x^0 に対応するビット1とビット0について，XORを取ります．基本CRC回路と高速CRC回路の違いは，レジスタのMSBに対してデータ入力をXORするかどうかです．

● データのバス幅に合わせた回路のアレンジ

ここまでは，データが1ビットずつくるという前提で話を進めましたが，実際には8ビットずつ，16ビットずつというよう

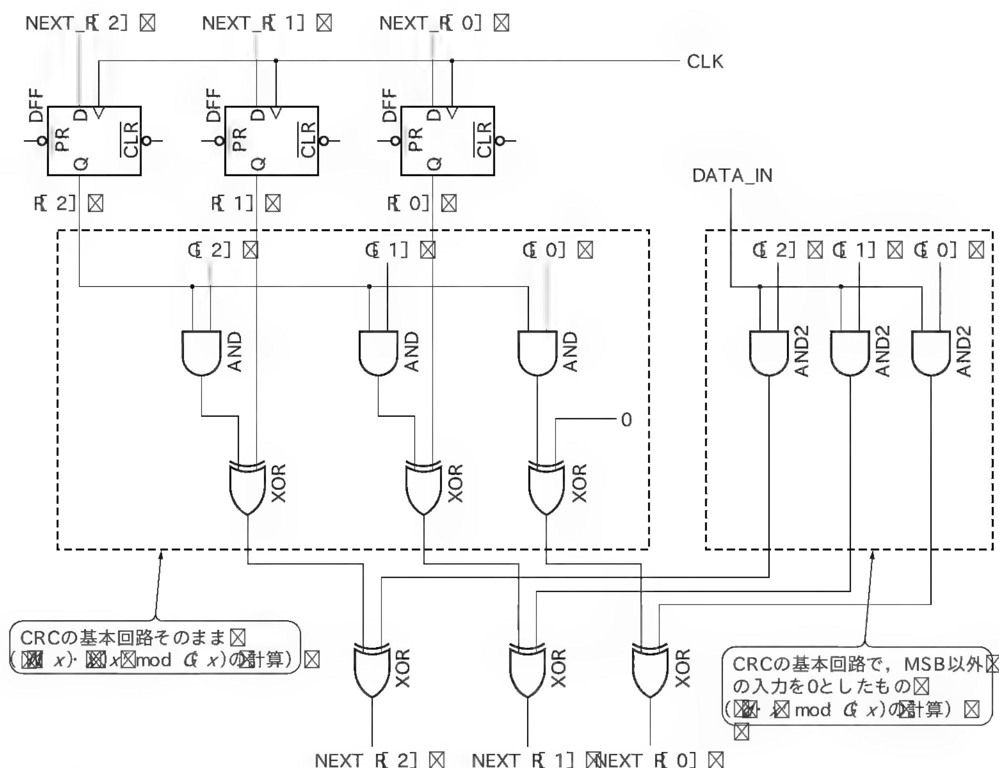


図9　高速CRCの実際の回路

図6の基本回路をベースに，送信時に0を投入しなくて済むよう改良した回路．任意の生成多項式が使える．ロジックはもう少し簡単化できる

に複数ビット単位でやってくることもあります．

その場合には，図6や図9，図10などで示した回路の組み合わせ回路部を直列につなげると（図11），一度に複数ビットを受け付けることができるようになります．なお，生成多項式を固定した場合，回路はXORゲートだけになりますが，（*A* xor（*B* xor（*C* xor *D*）））と直列につながったXORを（（*A* xor *B*）xor（*C* xor *D*））と木構造につなぎ直すことで，ゲート段数を減らして高速化できます．参考文献（2）や（5）を参照してください．

● CRCのバリエーションに応じた回路のアレンジと注意点

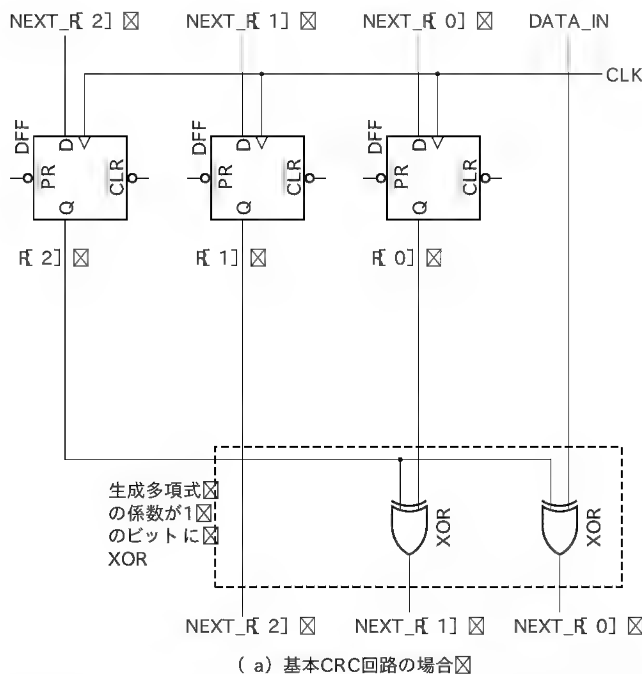
2節の最後でCRCのいろいろなバリエーションについて触れました．場合によっては，これまで説明してきた回路がそのままでは使えないことがあるので，注意点を次に説明します．非常に重要です．

● 注意1

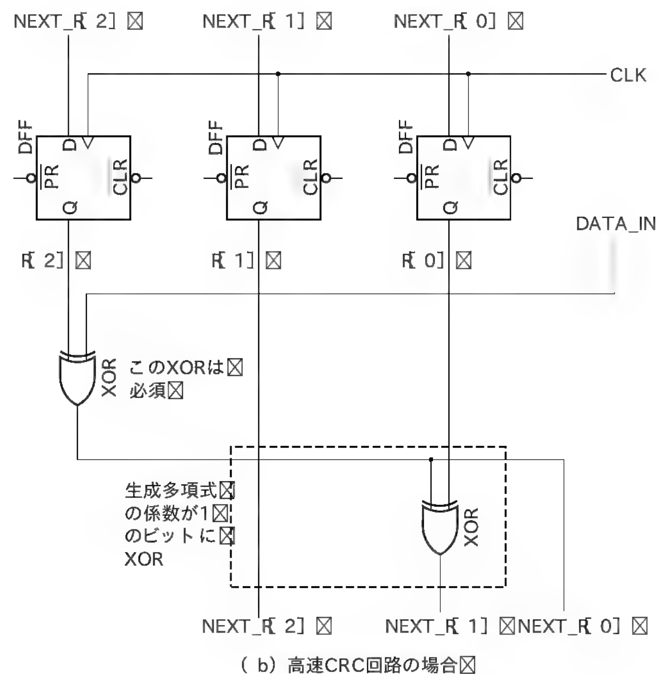
CRCのレジスタ初期値が0でない場合，3節の基本回路［図6と図10（a）］は使えません．送信側，受信側とも4節の高速回路［図9と図10（b）］を使ってください．当然のことですが，両者の初期値は揃っている必要があります．

● 注意2

パリティをMSB/LSB反転したり，またはパリティ定数値とのXORを取ったりしてから送信する場合，受信側において，データをそのまま除算して0判定してしまっては，うまくエラー検出できません．パリティを加工前の形に戻してから除算する



（a）基本CRC回路の場合



（b）高速CRC回路の場合

図10　生成多項式を固定すると回路を簡単化できる（生成多項式 x^3+x+1）
これらは，教科書などで紹介されている回路と同じものである

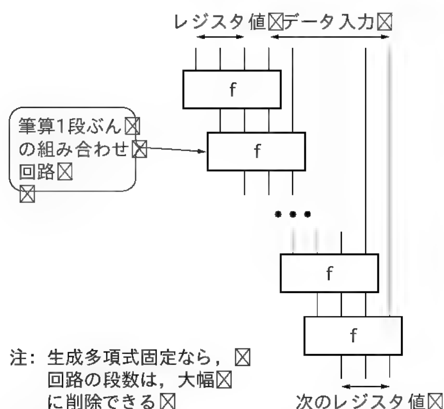


図11　パラレルにCRCを計算
組み合わせ回路部を複数個直列につなげれば，データ入力をパラレル化するとともに，少ないクロック数で処理できる



図12　サンプルCRC回路（リスト1）の入出力タイミング

リスト 1　汎用高速 CRC の HDL 記述例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity crc16_general is
    port (
        data_in     : in  std_logic;       -- input data value
        shift_in    : in  std_logic;       -- shift data
        init_in     : in  std_logic;       -- init data
        init_val    : in  std_logic_vector(15 downto 0);
                                           -- init value
        final_val   : in  std_logic_vector(15 downto 0);
                                           -- final XOR value
        genpoly_val : in  std_logic_vector(15 downto 0);
                                      -- generator polynomial
        reset_n     : in  std_logic;       -- reset (active low)
        clk         : in  std_logic;       -- system clock
        data_out    : out std_logic_vector(15 downto 0)
                                           -- CRC output
    );
end crc16_general;

architecture RTL of crc16_general is

    function fast_round(prev_parity : std_logic_vector(
                                              15 downto 0);
                     data    : std_logic;
                     genpoly : std_logic_vector(15 downto 0))
            return std_logic_vector is
        variable d_bit    : std_logic;
        variable d_val    : std_logic_vector(15 downto 0);
    begin
        d_bit    := prev_parity(15) xor data;
        for I in 0 to 15 loop
            d_val(I)     := d_bit;
        end loop;
        return ((genpoly and d_val) xor (prev_parity(14 downto
                                                  0) & '0'));
    end fast_round;

    signal  parity_reg  : std_logic_vector(15 downto 0);

begin

    u0: process (clk, reset_n) begin
        if (reset_n = '0') then
            parity_reg  <= (others => '0');
        elsif (clk'event and clk = '1') then
            if (init_in = '1') then
                parity_reg  <= init_val;
            elsif (shift_in = '1') then
                parity_reg  <= fast_round(parity_reg, data_in,
                                                  genpoly_val);
            end if;
        end if;
    end process;

    data_out    <= parity_reg xor final_val;

end RTL;
```

（ a）VHDL 記述例

```
module crc16_general(data_in, shift_in, init_in, init_val,
                 final_val, genpoly_val, reset_n, clk, data_out);
    input           data_in;    // input data value
    input           shift_in;   // shift data
    input           init_in;    // init data
    input   [15:0]  init_val;   // init value
    input   [15:0]  final_val;  // final XOR value
    input   [15:0]  genpoly_val;// generator polynomial
    input           reset_n;    // reset (active low)
    input           clk;        // system clock
    output  [15:0]  data_out;   // CRC output

    function [15:0] fast_round;
        input   [15:0]  prev_parity;
        input           data;
        input   [15:0]  genpoly;
    begin
        fast_round  = (genpoly & {16{prev_parity[15] ^ data}})
                        ^ ({prev_parity[14:0], 1'b0});
    end
    endfunction

    reg     [15:0]  parity_reg;

    always @(posedge clk or negedge reset_n) begin
        if (reset_n == 1'b0) begin
            parity_reg  <= {16{1'b0}};
        end else begin
            if (init_in == 1'b1) begin
                parity_reg  <= init_val;
            end else if (shift_in == 1'b1) begin
                parity_reg  <= fast_round(parity_reg,data_in,
                                                  genpoly_val);
            end
        end
    end

    assign  data_out    = parity_reg ^ final_val;

endmodule
```

（ b）Verilog HDL 記述例

か，データ部だけの CRC を計算して受け取ったパリティと比較する，というような回路にする必要があります．

● 注意3

CRC を使う場合，普通はデータが多項式の高次数側から送られてくるはずですが，もし逆順に送られてくるとうまく計算できません．一度バッファリングしてビット順を反転してから除算します．

5 HDL による回路実装例

● いろいろな HDL 記述例

これまでに示したいろいろな CRC 回路を HDL で記述してみましょう．

まず，パリティ長 16 ビットの汎用 CRC（図 9）の VHDL，Verilog HDL によるコーディング例をリスト 1 に示します．動作のタイミング・チャートは図 12（p.189）のとおりで，レジスタを初期化してからデータを投入していき，最終データの次のクロックで計算結果が出てきます．生成多項式は切り替え可能で，15次～0次の係数値を入力 genpoly_val に与えます．リスト 1 は図 9 の回路を HDL で記述しているだけで，とくにこれといったくふうはありません．筆算 1 段分の回路を function にしていますが，子モジュールにしてインスタンシエーションしてもかまいません．

次に，図 10 のように生成多項式を固定した場合の HDL 記述例（function 部のみ抜粋）をリスト 2 に示します．しかし，このような最適化を手でやらず，リスト 1 に示した function の

リスト 2　生成多項式を固定した場合の HDL 記述例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity crc_ccitt is
    port (
        data_in     : in  std_logic;        -- input data value
        shift_in    : in  std_logic;        -- shift data
        init_in     : in  std_logic;        -- init data
        init_val    : in  std_logic_vector(15 downto 0);
                                            -- init value
        final_val   : in  std_logic_vector(15 downto 0);
                                            -- final XOR value
        reset_n     : in  std_logic;        -- reset (active low)
        clk         : in  std_logic;        -- system clock
        data_out    : out std_logic_vector(15 downto 0)
                                            -- CRC output
    );
end crc_ccitt;

architecture RTL of crc_ccitt is

    function fast_round(prev_parity : std_logic_vector
                                (15 downto 0); data : std_logic)
              return std_logic_vector is
        variable add_data    : std_logic;
        variable next_parity : std_logic_vector(15 downto 0);
    begin
    -- CRC-CCITT    x^16+x^12+x^5+1
        add_data        := prev_parity(15) xor data;
        next_parity(0)  := add_data;
        next_parity(1)  := prev_parity(0);
        next_parity(2)  := prev_parity(1);
        next_parity(3)  := prev_parity(2);
        next_parity(4)  := prev_parity(3);
        next_parity(5)  := prev_parity(4) xor add_data;
        next_parity(6)  := prev_parity(5);
        next_parity(7)  := prev_parity(6);
        next_parity(8)  := prev_parity(7);
        next_parity(9)  := prev_parity(8);
        next_parity(10) := prev_parity(9);
        next_parity(11) := prev_parity(10);
        next_parity(12) := prev_parity(11) xor add_data;
        next_parity(13) := prev_parity(12);
        next_parity(14) := prev_parity(13);
        next_parity(15) := prev_parity(14);
        return next_parity;
    end fast_round;

    signal  parity_reg  : std_logic_vector(15 downto 0);

begin

    u0: process (clk, reset_n) begin
        if (reset_n = '0') then
            parity_reg  <= (others => '0');
        elsif (clk'event and clk = '1') then
            if (init_in = '1') then
                parity_reg  <= init_val;
            elsif (shift_in = '1') then
                parity_reg  <= fast_round(parity_reg, data_in);
            end if;
        end if;
    end process;

    data_out    <= parity_reg xor final_val;

end RTL;
```

(a) VHDL 記述例

```
module crc_ccitt(data_in, shift_in, init_in, init_val,
                 final_val, reset_n, clk, data_out);
    input           data_in;    // input data value
    input           shift_in;   // shift data
    input           init_in;    // init data
    input   [15:0]  init_val;   // init value
    input   [15:0]  final_val;  // final XOR value
    input           reset_n;    // reset (active low)
    input           clk;        // system clock
    output  [15:0]  data_out;   // CRC output

    function [15:0] fast_round;
        input   [15:0]  prev_parity;
        input           data;
        reg             add_data;
        reg     [15:0]  next_parity;
    begin
        // CRC-CCITT    x^16+x^12+x^5+1
        add_data    = prev_parity[15] ^ data;
        next_parity[0]  = add_data;
        next_parity[1]  = prev_parity[0];
        next_parity[2]  = prev_parity[1];
        next_parity[3]  = prev_parity[2];
        next_parity[4]  = prev_parity[3];
        next_parity[5]  = prev_parity[4] ^ add_data;
        next_parity[6]  = prev_parity[5];
        next_parity[7]  = prev_parity[6];
        next_parity[8]  = prev_parity[7];
        next_parity[9]  = prev_parity[8];
        next_parity[10] = prev_parity[9];
        next_parity[11] = prev_parity[10];
        next_parity[12] = prev_parity[11] ^ add_data;
        next_parity[13] = prev_parity[12];
        next_parity[14] = prev_parity[13];
        next_parity[15] = prev_parity[14];

        fast_round = next_parity;
    end
    endfunction

    reg     [15:0]  parity_reg;

    always @(posedge clk or negedge reset_n) begin
        if (reset_n == 1'b0) begin
            parity_reg  <= {16{1'b0}};
        end else begin
            if (init_in == 1'b1) begin
                parity_reg  <= init_val;
            end else if (shift_in == 1'b1) begin
                parity_reg  <= fast_round(parity_reg,data_in);
            end
        end
    end

    assign  data_out    = parity_reg ^ final_val;

endmodule
```

(b) Verilog HDL 記述例

引き数に生成多項式を定数として与え，EDA ツールに論理簡単化をやらせても実用上はあまり問題ありません(データ幅が広い場合を除く)．そうすることで，生成多項式のパラメータ化が可能になるというメリットもあります．

最後に，図 11 のように回路をつないでデータ入力をパラレルにした記述例を，リスト 3に示します．function 内で for ループを回すことで，1段ぶんの回路が直列につながります．また，データ入力ビット数をパラメータ化しておくと再利用に便利です．

なお，CRC を使う多くの場合において，モジュールの入出力インターフェースをデータ入出力タイミングに応じていろいろ作り変える必要があります．CRC 計算本体よりも，そちらのほ

うがずっとめんどうかもしれません．これについては参考文献(2)に多くの例があるので，参照してください．

● 回路の実装性能

CRC 演算器は回路のゲート段数が少ないため，現在のASIC/FPGA デバイスではきわめて高速なクロックで動かすことが可能です．

図13にFPGA(Stratix II)にリスト3の回路をマップした場合の動作周波数を示します．ここでは，データ幅のぶんだけ演算器が直列につながる回路構成をとったため，データ幅nに対してクリティカル・パス遅延は$O(n)$となります(つまり比例)．しかし，4節でも触れましたが，生成多項式を固定してXORゲートの接続を木構造の形に修正すれば，データ幅nに対して$O(\log n)$のクリティカル・パス遅延に抑えることはできます．

まとめると，データ幅が8ビットや16ビット程度であれば，わかりやすさを優先した回路の組み方・HDLの記述スタイルを用いても，現在のデバイス能力なら性能上の問題はほとんど

リスト3　入力をパラレル化したHDL記述例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity crc16_general_multi is
    generic (
        ROUND_NUM   : integer   := 8
    );
    port (
        data_in     : in  std_logic_vector
                    (ROUND_NUM - 1 downto 0); -- input data value
        shift_in    : in  std_logic; -- shift data
        init_in     : in  std_logic; -- init data
        init_val    : in  std_logic_vector(15 downto 0);
                              -- init value
        final_val   : in  std_logic_vector(15 downto 0);
                              -- final XOR value
        genpoly_val : in  std_logic_vector(15 downto 0);
                              -- generator polynomial
        reset_n     : in  std_logic; -- reset (active low)
        clk         : in  std_logic; -- system clock
        data_out    : out std_logic_vector(15 downto 0)
                              -- CRC output
    );
end crc16_general_multi;

architecture RTL of crc16_general_multi is

    function fast_round(prev_parity : std_logic_vector
                                          (15 downto 0);
                        data    : std_logic_vector
                                      (ROUND_NUM - 1 downto 0);
                        genpoly : std_logic_vector(15 downto 0))
            return std_logic_vector is
        variable tmp_result : std_logic_vector(15 downto 0);
        variable d_bit       : std_logic;
        variable d_val       : std_logic_vector(15 downto 0);
    begin
        tmp_result  := prev_parity;
        for I in ROUND_NUM - 1 downto 0 loop
            d_bit   := tmp_result(15) xor data(I);
            for J in 0 to 15 loop
                d_val(J)    := d_bit;
            end loop;
            tmp_result  := ((genpoly and d_val) xor
                                (tmp_result(14 downto 0) & '0'));
        end loop;
        return tmp_result;
    end fast_round;

    signal  parity_reg  : std_logic_vector(15 downto 0);

begin

    u0: process (clk, reset_n) begin
        if (reset_n = '0') then
            parity_reg  <= (others => '0');
        elsif (clk'event and clk = '1') then
            if (init_in = '1') then
                parity_reg  <= init_val;
            elsif (shift_in = '1') then
                parity_reg  <= fast_round(parity_reg,
                                   data_in, genpoly_val);
            end if;
        end if;
    end process;

    data_out    <= parity_reg xor final_val;

end RTL;
```

(a) VHDL記述例

```
module crc16_general_multi(data_in, shift_in, init_in,
      init_val, final_val, genpoly_val, reset_n, clk, data_out);
    parameter       ROUND_NUM   = 8;

    input   [ROUND_NUM - 1:0]   data_in;  // input data value
    input           shift_in;        // shift data
    input           init_in;         // init data
    input   [15:0]  init_val;        // init value
    input   [15:0]  final_val;       // final XOR value
    input   [15:0]  genpoly_val;     // generator polynomial
    input           reset_n;         // reset (active low)
    input           clk;             // system clock
    output  [15:0]  data_out;        // CRC output

    function [15:0] fast_round;
        input   [15:0]              prev_parity;
        input   [ROUND_NUM - 1:0]   data;
        input   [15:0]              genpoly;
        reg     [15:0]              tmp_result;
        integer I;
    begin
        tmp_result  = prev_parity;
        for (I = ROUND_NUM - 1; I >= 0; I = I - 1) begin
            tmp_result  = (genpoly &
                              {16{tmp_result[15] ^ data[I]}})
                          ^ ({tmp_result[14:0], 1'b0});
        end
        fast_round  = tmp_result;
    end
    endfunction

    reg     [15:0]  parity_reg;

    always @(posedge clk or negedge reset_n) begin
        if (reset_n == 1'b0) begin
            parity_reg  <= {16{1'b0}};
        end else begin
            if (init_in == 1'b1) begin
                parity_reg  <= init_val;
            end else if (shift_in == 1'b1) begin
                parity_reg  <= fast_round
                               (parity_reg,data_in,genpoly_val);
            end
        end
    end

    assign  data_out    = parity_reg ^ final_val;

endmodule
```

(b) Verilog HDL記述例

ないといえます．それ以上のデータ幅の場合は，専用ツールを作るか，または手作業をして，回路の構造を最適化するほうがよいでしょう．

● FPGAによる実装デモ

CQ出版(株)から発売されているStratix FPGAボード[(10)]にCRC回路の簡単なデモを実装し，動作させてみました．ボードとPCをRS-232-Cで接続し，ハイパーターミナルから16進数の列を打ち込むと，そこまでのCRC値がリアルタイムでLEDに表示されます．生成多項式も自由に設定可能です(写真1)．

ここで設計したデモ回路は，プロジェクト・ファイル一式のアーカイブを`http://www.cqpub.co.jp/interface/`からダウンロードできるようにする予定なので，Stratixキットをお持ちの方は一度お試しください．

まとめ

CRC計算回路はこれまでに多くの文献で紹介・説明されていますが，本稿のリスト2に示した「最終的な回路」だけが示されている場合がほとんどです．つまり，リスト2の回路でなぜ除算ができているのかは，あまり説明されないことが多いのです．しかし，応用で回路をいろいろアレンジしていくには，そこが一番重要なキーポイントです．本稿では，普通とは一味違った説明をしたつもりですが，皆さんのお役に立てば幸いです．

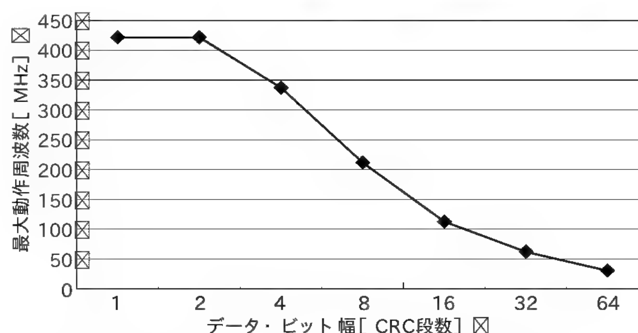


図13 FPGAにマップしたCRC回路の動作周波数(ALTERA Stratix II EP2S15F484C3，Quartus II ver4.1使用)
演算回路を直列につないだ場合，データ幅に比例して回路遅延は増える．ただし，生成多項式を固定すれば，データ幅のlogに比例する程度の回路遅延に抑えることは可能

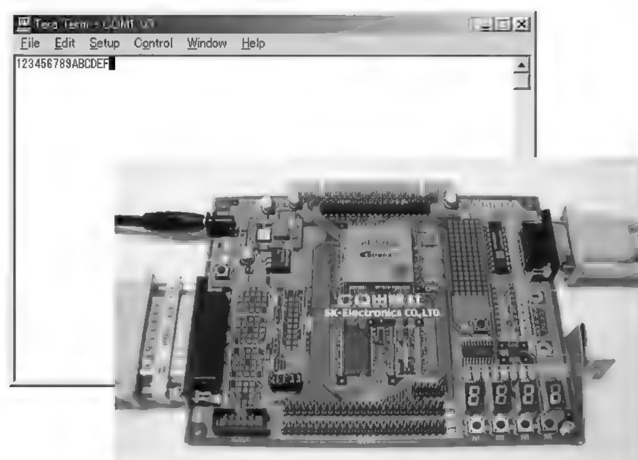


写真1 CRC計算のデモのようす

参考文献

(1) 森岡澄夫；“エラー訂正や暗号処理で使われる演算回路を究める”，デザインウェーブマガジン，2003年7月号，pp.57-67.
(2) 森岡澄夫；“HDLによる高性能ディジタル回路設計”，CQ出版社，2002.
(3) 今井秀樹；“符号理論”，電子情報通信学会，1990年
(4) 片山泰尚，森岡澄夫；“ハードウェアで高速処理を実現したリードソロモン復号アルゴリズム(上)”，Interface，2000年6月号，pp154-160.
(5) 片山泰尚，森岡澄夫；“ハードウェアで高速処理を実現したリードソロモン復号アルゴリズム(下)”，Interface，2000年7月号，pp164-170.
(6) `http://www.easics.be/webtools/crctool`(生成多項式をセットするとVerilogやVHDLのコードの一部を作ってくれる)
(7) `http://www.metreoncreations.com/crc.asp`(同上)
(8) `http://rcswww.urz.tu-dresden.de/~sr21/crc.html`(web版CRC計算機)
(9) `http://standards.ieee.org/getieee802/802.3.html`(イーサネットにおけるCRC仕様．IEEE 802.3)
(10) Stratix評価キット `http://www.cqpub.co.jp/eda/Stratix/default.htm`(FPGAボード)

もりおか・すみお `http://www02.so-net.ne.jp/~morioka/`

アナログ回路もプログラマブルな時代へ

プログラマブル・アナログIC "FPAA"のセンサ処理への応用

Steve Harrold

プログラマブル・アナログICの登場

CPLDやFPGAといった「プログラマブル・ディジタル・システム・オン・チップ(SoC)」部品の登場により，機器メーカは1枚の回路基板と一組の部品在庫で，多様な製品やインターフェースを作成できるようになった．プログラマブルSoCが事業に与える利点は，規模の経済によるコスト削減，高度に差別化された製品の早期市場投入など，数え切れないものがある．これらプログラマブルSoCの存在意義は，新設計アイデアの短期かつ簡便な試作だけに留まらず，急速に変化する市場での競争に打ち勝つための手段へと進化を遂げている．

プログラマブル・ディジタルSoC部品の弱点の一つは，取り扱うシステムのアナログ部分を(事実上)無視していたことだ．とくにセンサを含むシステムではアナログ回路部が必須であり，これをどのように実装するかは重要な課題とされていた．

たとえば，センサによって得られたアナログ信号をA-Dコンバータ(ADC)に入力する場合には，ADCの性能を十分に引き出すために，適切に前処理された入力信号をADCへ与える必要がある．

この信号前処理には二つの仕事がある．まず，入力信号の周波数帯域をADCのサンプリング周波数の2分の1以下に制限して，望ましくない高周波成分やノイズがエイリアシングにより低周波部に混入するのを防ぐこと，次にダイナミック・レンジを最大化してディジタル化された信号のノイズを最小にするために，ADCのフルスケール入力範囲に合わせて信号を増幅/レベル・シフトすることである．

一つの信号前処理回路を二つの顧客A社とB社に納入する場合を考えてみよう．ここでA社がX社のセンサを使用し，B社がY社のセンサ使用したいという要望があったとする．このような場合，それぞれのセンサの信号特性が異なることから，アナログ回路に何らかの修正が必要になるだろう．このため，従来は信号前処理のために2種類の基板が必要になっていた．しかし，プログラマブル・アナログICの進化のおかげで，今では同じチップで両者に対応することができる．

プログラマブル・パラメータの時代

「プログラマブル・アナログ」ということばは，人によって定義が異なる．ICメーカもまたプログラマビリティを提供するために多様な方法を提唱してきた．

もっとも簡単なものとしては，機能は固定で，パラメータのみプログラマブルなチップがアナログ仕様の微調整に用いられてきた．多くの回路は，(アンプのゲインや帯域幅，ローパス・フィルタの遮断周波数のような)バイアス電流に依存する性能パラメータをもっている．回路への電流供給のもとになる基準電流をプログラムすれば，これらのパラメータを制御できることになる．

基準電流をプログラムするためには，種々の手法が使われてきた．外付けの抵抗器によるごく簡単な方法や，D-Aコンバータ(DAC)により電流値を決めることでより正確な制御ができるものもある．

DACへの入力は，電源投入時にPROMからダウンロードしたり，あるいはマイコン上のプログラムで直接制御することができる．不揮発性とプログラマビリティを持たせるために，フローティング・ゲート・トランジスタを用いて基準電流を決めているものもある．この素子は電気的にしきい電圧をプログラムすることができ，しかも長期間，再度プログラムされるまで安定に保持することができる．

プログラマブル機能設定の時代

回路のパラメータだけでなく，機能も書き換える必要のある用途では，単なるプログラマビリティだけでなく機能書き換え性(コンフィギャラビリティ)を実現するために，より高度な手段が必要とされる．

たとえば，ある用途ではアンプの前にローパス・フィルタをおき，別の用途ではフィルタは不要だがアンプに高ゲインが必要なことがあるだろう．また，ローパス・フィルタの代わりにバンドパス・フィルタが必要な場合もある．

このような大幅な変更になると，単にバイアス電流を調整す



るだけでは対応できない．何らかの手段で，IC内の信号の流れを切り替える必要がある．これは，回路をビルディング・ブロック群から選び，プログラマブルなスイッチ（ないしはアンチ・ヒューズ）でブロック間の配線を指定することで可能となる．各ビルディング・ブロック自体も，（アンプ，抵抗，コンデンサ，スイッチなどの）部品の組み合わせでできている．

この手法により，増幅器（図1）はOPアンプと，その負側入力と信号源間の抵抗（入力系），その負側入力と出力間の抵抗（フィードバック系）で構築することができる．フィードバック系に並列抵抗を追加すると，この段のゲインが下がる．並列抵抗のかわりにコンデンサをフィードバック系に接続すると，ローパス・フィルタができあがる．ブロック間やブロック内のスイッチの開閉状態，ICの機能，およびその回路パラメータを，このように機能設定データに従って決定することができる．このデータは，一般的には電源投入時にPROMからダウンロードして与えられる．

ICメーカは，機能書き換え可能なプログラマブル・アナログICを作るためにさまざまな手法を用いてきた．もっとも簡単なものは，入力系とフィードバック系に従来のアナログ部品（抵抗とコンデンサのみ．コイルはマイクロ波領域でないと大きすぎて集積化できない）を用いたものだ．

この方式では，入力信号がつねに読み込まれ，出力信号がつねに有効であるため，いわゆる「連続系回路」になる．連続系回路は，ADCで信号がサンプリングされるときのエイリアシング効果を防止する前置フィルタ用に適している．

この方式の欠点は，フィルタのパラメータが一般的にRC時定数によって決まるため，IC内の抵抗やコンデンサの精度のバラツキの影響を受け，高精度で安定した動作を期待しにくいことである．この問題を緩和する方策の一つとして，時定数が抵抗値でなくアンプのトランス・コンダクタンスで決まるように回路を修正する方法がある．Lattice Semiconductor社はこの手法とコンデンサのトリミングを組み合わせ，ispPACチップで誤差±5%未満のフィルタを実現している．

入力系とフィードバック系の部品にダイオードを加えると，出力に入力の対数や逆対数成分を含むビルディング・ブロックを作り出すことができる．Zetex社は，これを活用して，信号処理目的の書き換え可能プログラマブル・アナログICであるTRACを作り出した．これにより乗除算のような複雑な仕事が，対数領域における単純な加減算に置き換えられる．時定数設定のための外部部品を追加すれば，微分や積分も組み込むことができる．

回路内の時定数の精度と安定度は，スイッチト・キャパシタ法の採用により大幅に改善できる．この手法の原理を図2に示す．スイッチはクロック信号（f_c）により開閉され，コンデンサが入力信号による充電と放電を交互に繰り返す．回路電流の平均値は$I=V \cdot f_c \cdot C$であり，$R=1/f_c \cdot C$の抵抗をつないだときの電流に等しい．言い換えれば，クロック周波数が信号周波数より十分に高ければ，スイッチト・キャパシタ回路は抵抗の役割を果たすことができる．

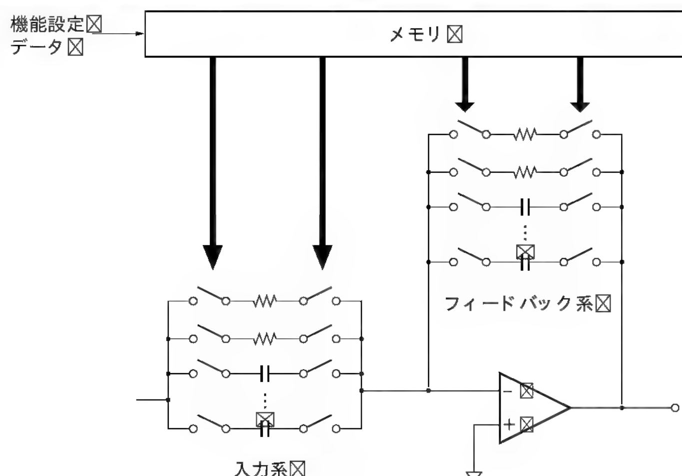


図1　プログラマブル・アナログICの書き換え可能ビルディング・ブロック
回路パラメータや機能の変更は，メモリ内のデータでスイッチの動作設定を変更することにより行われる

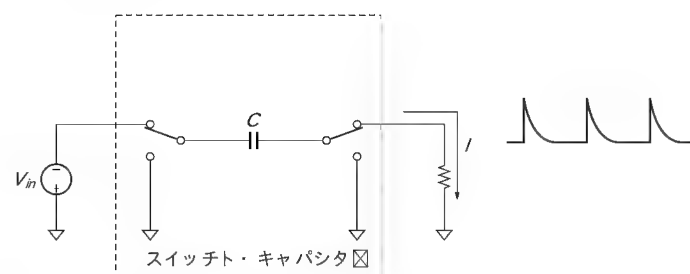


図2　スイッチト・キャパシタ回路の動作図
クロック・サイクルごとにコンデンサが入力信号による充電と負荷への放電を繰り返す．負荷への平均電流値は，抵抗値$R=(f_c \cdot C)^{-1}$の抵抗を介した場合と等しい

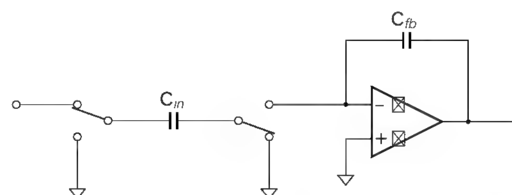


図3　スイッチト・キャパシタ積分器
積分時定数はコンデンサの容量比で決まり，誤差1%以下になる．この定数は電源電圧や温度の変化に左右されにくく，経時変化もない

この手法は一見たいしたメリットもなく，回路を複雑にするだけに見えるかもしれない．しかしながら，OPアンプとコンデンサを組み合わせると，積分定数が$f_c \cdot C_{in}/C_{fb}$で決まる積分器を作ることができる（図3）．すなわち，積分定数が二つのコンデンサの容量比で決まることになるが，この比率からICチップ上では誤差1%以下（typ.: 0.1%）という高精度を実現できる．しかも抵抗器の代わりにスイッチト・キャパシタを用いた

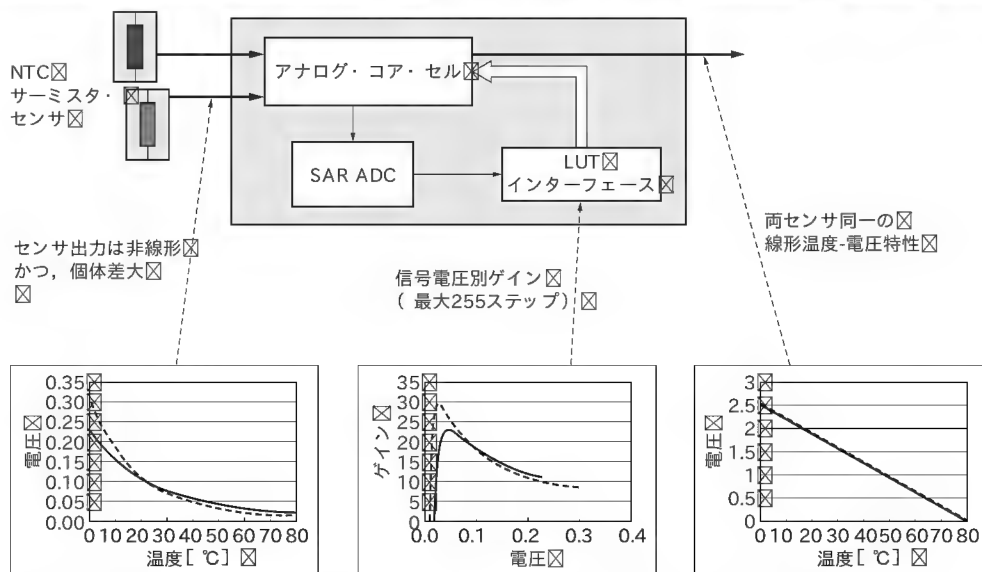


図4　ダイナミック機能書き換えによるセンサ信号の線形補正
アナログ増幅器のゲインを入力信号レベルに応じて変更し線形化．センサの個体差にも，補正データの変更で対応できる

増幅器は，同等のゲイン精度を示すうえに，入力側コンデンサのスイッチング・タイミングを変えるだけで極性切り替えができるという利点もある．

実際，スイッチト・キャパシタ方式を使えば，さまざまな機能をもつビルディング・ブロックを簡単に作り出すことができ，しかもその回路動作がコンデンサの容量比で決まるため，電源電圧や温度，あるいは経時変化の影響をほとんど受けない．ビルディング・ブロックの例には，加減算増幅器，発振器，フィルタ，整流器，積分器，微分器などがある．容量比は，入力系とフィードバック系のコンデンサの値により簡単に変更できる．

この方式の弱点は，入力信号がスイッチト・キャパシタでサンプリングされるために，ADC同様にエイリアシングを受ける可能性があるということである．このため，サンプリング周波数の2分の1以上の周波数成分がスイッチト・キャパシタ回路に入り込まないように連続系の前置フィルタが必要になる．

しかし，スイッチト・キャパシタによる等価抵抗を有効にするため，サンプリング周波数は一般的に信号周波数よりはるかに高くされており，前置フィルタは*RC*フィルタなどの簡単なものでよい場合が多い．

FPAAによるダイナミック機能書き換えの時代へ

現時点でのもっとも進んだプログラマブル・アナログICは，その機能と回路パラメータの両方を，動作開始時だけでなく動作中にも瞬時に変更できるというものだ．この「ダイナミック書き換え能力」によって，特定のセンサの特定の要求に応えたり，センサ素子の温度変化や経時変化にも適応できるアナログ回路システムを構築することができるようになる．

このレベルまでのプログラマビリティを持つICの一例が，アナダイム社のFPAA（Field Programable Analog Array）である．アナログ部の機能書き換えは，FPAA内部からの指示や，外部のCPUやMCUからの指示によって行われる．

たとえば，内蔵ADCと参照テーブル（LUT）でフィードバック・コンデンサの容量を書き換えて，増幅器のゲインを制御するようなこともできる．これにより非線形なセンサ信号を，信号レベルに応じたゲインで増幅して線形に補正することができる（図4）．

外部からの制御でFPAAを書き換える場合，書き換え用データはあらかじめ準備しておくことも，あるいは何らかのシステム・アルゴリズムに基づいてリアルタイムに作り出すこともできる．どちらの場合についても，書き換えデータ作りを支援するソフトウェアが利用できる．

再機能設定の場合，チップ全体の書き換えは不要で，内部アナログ・ブロックのパラメータ数個を書き換えるだけで済むことが多い．このため，FPAAはチップのほかの部分が普通に動作している最中に，回路の一部のみ書き換えることを可能にしている．この「部分書き換え」は通常は数μsで完了する．

FPAAのダイナミック機能書き換えにより，複数の固定機能のハードウェア・ブロックを1個のプログラマブルなハードウェアに置き換えることができる．この「ハードウェア・マルチプレクシング」は，たとえば電話のダイヤル音検出器やマルチセンサ用のデータ・ロギング・インターフェースに利用できる．前者の例（図5）では，ダイヤル音検出用に多くの個別フィルタを並べる代わりに，一個のフィルタに順次異なるフィルタ・パラメータを書き込む．後者の例（図6）では，複数のセンサを個



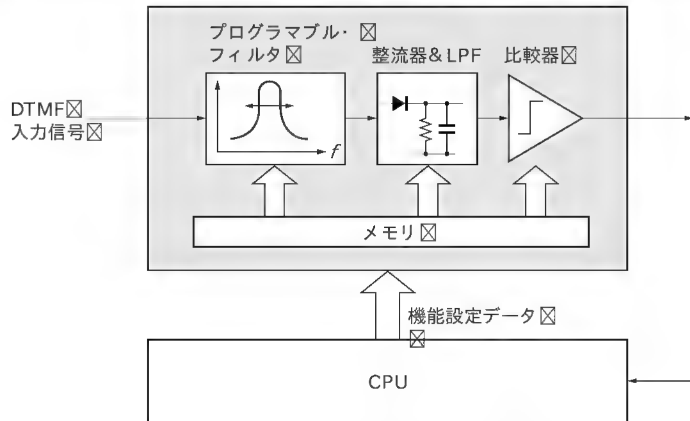


図5　ハードウェア・マルチプレクスされたダイヤル音検出器
CPUがフィルタ特性を順次切り替える．ダイヤル音を検出するごとに，比較器出力が"H"レベルになる．国ごとに異なる周波数基準にも，機能設定データ変更だけで簡単に対応できる

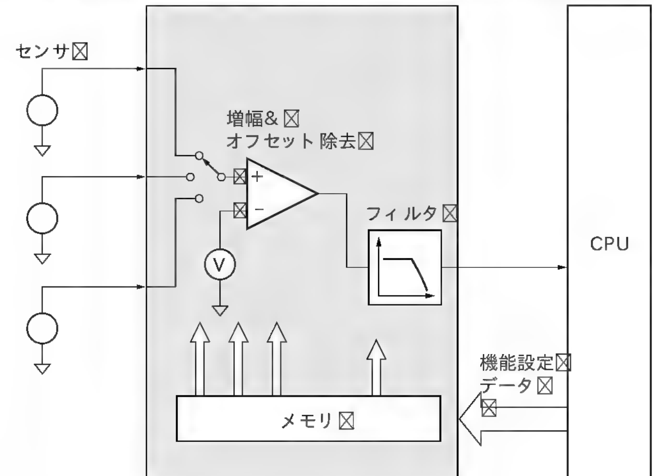


図6　マルチセンサ・データ・ロガー
センサごとに信号前処理回路を最適化するために，データ記録に先立って最適な機能設定データを書き込む．センサ感度低下などの経時変化なども容易に補正できる

別の入力ピンに接続し，入力部を切り替えつつセンサごとに異なる要求事項に応じたアナログ処理を施すように，順次機能を書き換える．この能力により，大幅な基板面積とコストの削減が図れる．

おわりに

プログラマブル・アナログICは，別々の製品を共通の基板で組み立て，別々の性能や機能が求められる場面においてコストを削減する魅力的な手段を与えてくれる．これらのICは設計技術者に新たなパラダイムを提供する．センサ信号はプログラマブル・ディジタルICの柔軟性という利点を活用するために，しばしば可能な限り前段部でディジタル化されてきた．いまや，この柔軟性は，高価なADCやDSPチップに置き換えることなくアナログの世界で享受できる．たった1枚の基板を複数のアプリケーションや顧客向けにプログラムできるようになり，ディジタルの世界同様にアナログの世界でもフレキシブルかつ高精度に機能と性能を発揮させる時代が到来した．

Steve Harrold　Anadigm社 アナログIC技術マネージャ
翻訳：浦崎 一明　オムロン(株) セミコンダクタ統括事業部

モータ制御にART-Linuxを活用した

Linuxによる ロボット・アームのリアルタイム制御

吉川 智康



ロボット制御に必要な技術

● ロボットの制御って？

ロボットといっても，分解してみれば各関節を駆動している複数のモータが組み合わさっているだけにすぎません．つまりロボット制御は，複数のモータを並列に制御することと言い換えることができます（図1）．

ここでモータを1個だけ制御する場合を考えてみます．制御器側は，モータの回転速度または駆動電流を観測し，必要な制御演算を行い，駆動出力値を決めます．制御演算には，サンプリング時間（制御周期）が一定であることが求められます．制御対象モータの時定数にもよりますが，制御のためのサンプリング時間は，数百μsから数msの範囲で行われることが多いようです．

一つのCPUですべてのモータを制御する方法もあります．また，一つのモータ・ドライバに対して一つのCPUを配し，CPU間の通信を行う方法や，DSPをアシストとして使って複数のモータを制御する方法など，さまざまな方法があります．その場合でも数msから数十msでのサンプリング周期でモータの制御を行っています．

また，ロボット制御の場合，各関節を駆動するモータ制御のほかに，作業空間内での各関節の位置を求める演算（運動学計算）を行う場合があります．ロボットを動作させるための制御理論によっては，各関節速度から逆ヤコビアン行列を演算する必要があるかもしれません．

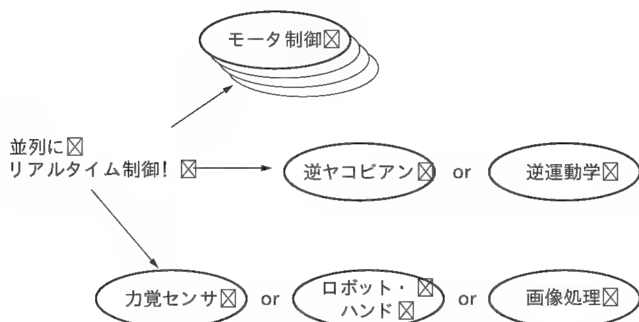


図1　ロボット制御に必要なリアルタイム制御

実際には，このような制御演算をさせる場合，固定小数点演算にしたり，演算結果をあらかじめテーブル化しておくことで，処理を高速化するくふうが行われています．

しかし近年，CPUの演算速度が急速に向上したことにより，一つのCPUですべてのモータを制御し，さらに余ったリソースでほかの制御機器も並列に制御させることが可能となってきました．

● DOSからの脱却

DOSは16ビット処理系です．メモリ空間640Kバイトの壁などもあり，大きいメモリ領域を扱うプログラムの作成は苦手としています．しかし，制御プログラムを書く場合，シングル・タスクであることは大きなメリットがあります．制御周期の設計にあわせたループ待ちなどを入れれば，安定した制御周期でループを実現できます．

このような理由で，依然として制御系のプログラムが，DOS環境から抜け出せないでいる場合も多いと聞きます．

● ロボット制御にWindows？　Linux？

しかし近年，CPUの高速化に合わせて複数の処理（タスク）を同時に行わせたいという要望が強くなってきました．とくにOSがグラフィカル・ユーザ・インターフェース（GUI）をもつようになると，マルチタスク化は必須になりました．

マルチタスクOSの場合，実行している各タスクへのCPU割り当てをOSが一括で管理する，プリエンプティブ・マルチタスクOSがほとんどになってきました．

ロボット制御の場合でも，マルチタスク化への要求は強くあります．たとえば，ロボット・アームとロボット・ハンド，画像処理，力覚センサなどとの並列制御などです．

タスクに対するCPUの割り当て時間をタイム・スライス値と呼びます．WindowsやLinuxでは10msです（Linuxカーネル2.6では1ms）．それでも，10msごとに安定してタスクが呼ばれればよいのですが，残念ながらそうはならないようです．実際には，タスクの優先度やほかの割り込み制御などにより，変動が大きく，安定したサンプリング周期は作れません．また，メカトロニクス制御の場合は，リアルタイム性を確保するため，



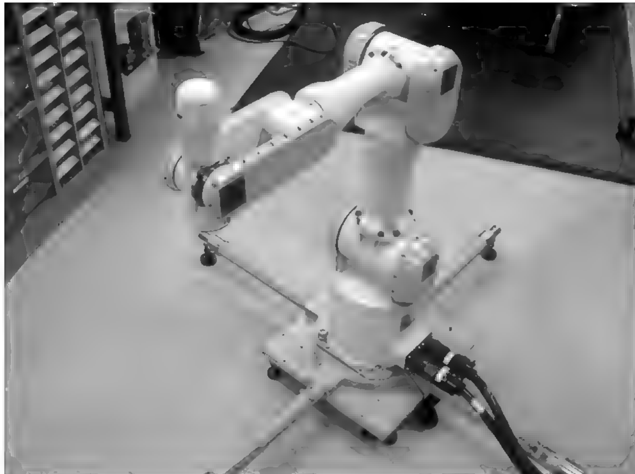


写真1　ロボット・アーム PA10(三菱重工業(株)神戸造船所)

表1　ロボット・アーム PA10の仕様

アーム長	1345mm
アーム重量	35kgf
手先可搬重量	10kgf
関節数	7
駆動	DCブラシレス・モータ
位置決め精度	± 0.1m
関節位置検出	出力軸ブラシレス・レゾルバ

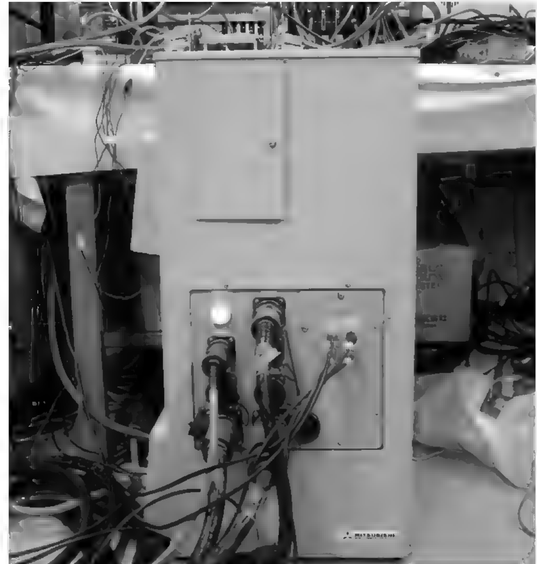


写真2　サーボ・ドライバ

表2　サーボ・ドライバの仕様

構成	セミディジタル・サーボ・ドライバ
指令値	速度指令，トルク指令
通信インターフェース	ARCNET
電源	AC200V(PA10-Cシリーズは100V)

もっと細かいサンプリング周期が必要となります．

そこで，リアルタイム性を保証したOSである，リアルタイムOSが必要となってくるわけです．リアルタイムOSに関しては，後述します．

制御対象のロボット・アームPA10とは

● オープンなロボット

ロボットに限らず，メカトロニクス制御を行おうと考えた場合，まずはハードウェアを自作するか市販のハードウェアを使うかの選択になります．自作の場合は，パーツ選定から組み立て，制御ソフトウェアの開発まで自分で行う必要がありますが，設計の自由度があります．ただ，高い精度を出すためには，ある程度の経験や工作の設備が必要にもなります．

市販のものは，自作のような苦労が少ないことは確かです．しかし，制御部分がブラックボックスになっていることがほとんどです．市販のロボット・アームの場合，手先位置などを数点コントローラに入力(この作業をティーチングと呼ぶ)します．その入力位置間の軌道などはコントローラに任せしてしまう使い方がほとんどです．この部分を改良して使うのは困難です(大学のロボット制御研究の場合，制御部分だけを自作したいという要望が高い！)．一部でRS-232-Cなど低速通信で制御できるロボット・アームも市販されているようですが，要求を満たしているとはとても言えませんでした．

これらのことを踏まえて，1993年，三菱重工(株)神戸造船所から発表になったロボット・アームがPA10です(写真1，表1)．

● ロボット・アームPA10の仕様

PA10の特徴は，次の点が挙げられます．

▶ オープン・システムを採用

▶ 7自由度をもつ(後に6自由度のアームもラインナップに追加)

▶ 軽量(35kgf)・高可搬重量(10kgf)

▶ 速度制御・トルク制御の切り替えが可能

PA10は機能を階層化し，階層ごとの規格を公開することで，ユーザごとに異なる利用方法を網羅しようという考え方で設計されました．つまり工場など，すでに動作パターンの決まったことを行わせる場合には，上位層を使って，ほかの市販ロボットと同じようにティーチングで制御することができます．また，ロボットの制御プログラムを自作したいユーザの場合は，低レベルのレイアを使って自作プログラムから制御することも可能です．この場合，サーボ・ドライバ(写真2，表2)がモータに対して出力する値を直接入力するようにします．

この低レベルのレイアでロボットのサーボ・ドライバを直接アクセスするには，ARCNETというネットワークを経由して行う仕様になっています．

また，PA10は，7自由度をもつロボット・アームです．この自由度の数は，人間の腕と同じです．図2に示した向きに各関節を駆動します(図3)．

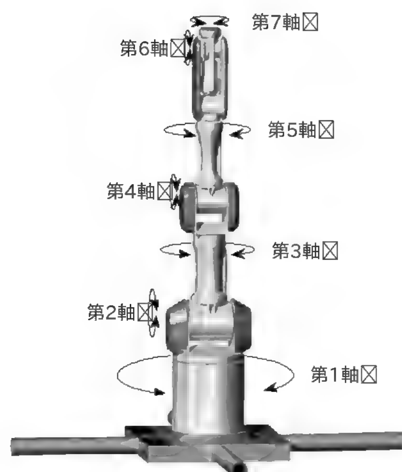


図2　ロボット・アーム PA10の関節の動き

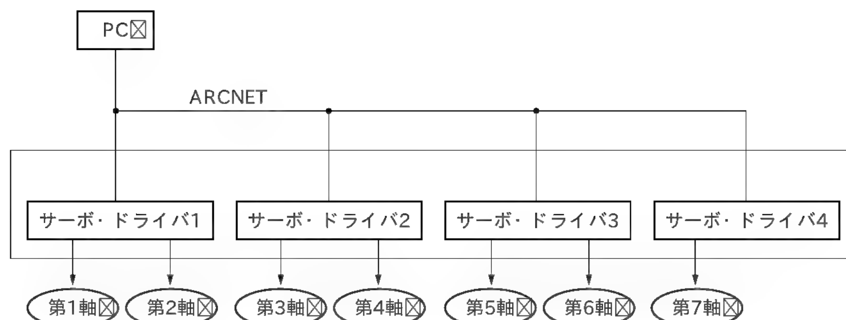


図3　ロボット・アーム PA10のブロック図

0	1	2	3	…	n	…	255
送信先ID	送信元ID	データ・オフセット(n)	Null	…	先頭データ	…	最終データ

図5　ARCNETのパケットの構造

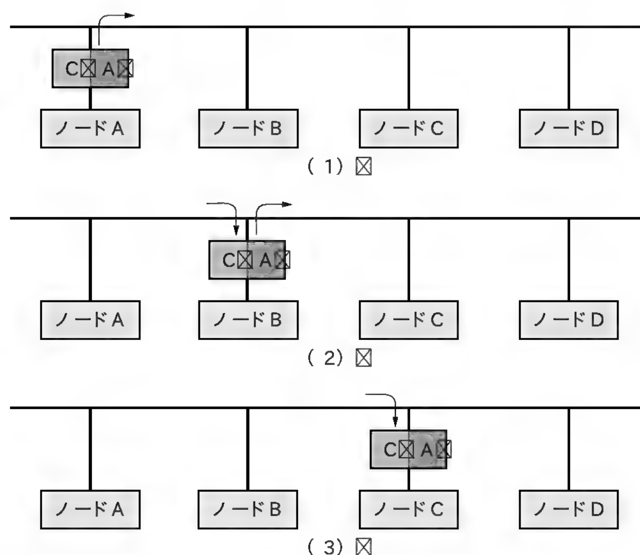


図4　ARCNETの概念図

ネットワーク規格 ARCNET での通信

● リアルタイム通信の必要性

RS-232-Cをはじめとするシリアル通信は，古くから使われています．現在でも，組み込み向けCPUには最初からその機能が内蔵されているものが多く出回っています．シリアル通信は1対1の通信であり，ノード間での通信速度があらかじめ決まっています．したがって，通信するデータ・サイズが決まれば，通信にかかる時間も決まります．つまり，リアルタイム性があるといえます．

しかし，リアルタイム性があるといっても，さすがに通信速度が遅いという点は否めません．サイズの大きいデータを送るには適していないといえます．そこで，リアルタイム性をもち，かつ高速通信が可能なデバイスとプロトコルの登場が待たれました．

● ARCNETでの通信

ARCNETは1977年に米国Datapoint社によって提唱された改良型トークン・パッシング・プロトコルのネットワーク規格です．発表された当初は，8インチのフロッピ・ディスク・ドライブの通信用だったそうです．OSI基本参照モデルの物理層，およびデータ・リンク層の一部がプロトコルとして実装されています．

トークン・パッシングとは，一つのトークンをノード間で巡回させるネットワークの通信方式です(図4)．トークンにパケットを乗せてデータ通信を行います．ARCNETの場合，トークンで運ばれるパケットの長さが一定です．したがってノードの数が一定であれば，ノード間のデータ通信速度が確定します．これにより，データ通信のリアルタイム性を保証しています．通信速度は標準で2.5Mbps，最大5Mbpsです．

また，パケット長は，ロング・パケット(512バイト長)とショート・パケット(256バイト長)が選択できます．

パケットに入るデータは図5に示すとおりです．まず先頭に送信先のID番号，次に送信元(自分自身)のID番号が入ります．送信データはパケットの後から詰まるように並べます．そのため，送信元IDの次のデータには，データ本体の先頭までのオフセット値が入る構造になっています．

現在，スタンダードマイクロシステムズ(株)より，ワン・チップARCNET用コントローラとしてCOM20020Dが発売されています．COM20020DにはARCNETのプロトコルが実装されているほか，2Kバイトの内蔵RAMがあります．また，COM20020Dを使ったPCIバスやISAバス対応のネットワーク・カード(写真3)も販売されています．



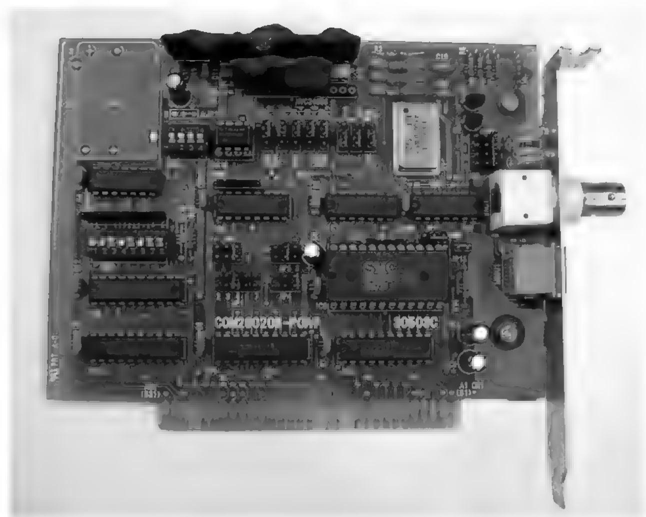


写真3　ARCNETネットワーク・カード(ISAバス)

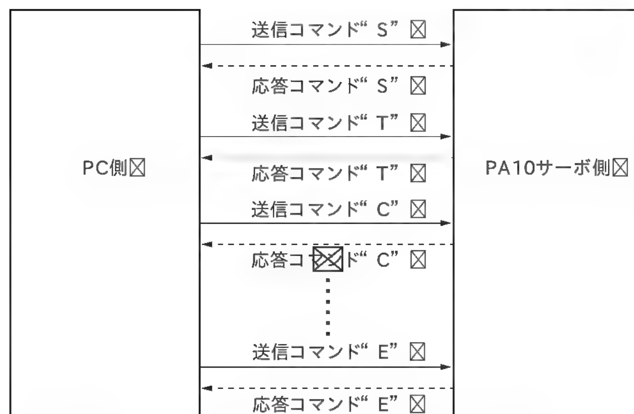


図6　ロボット・アームPA10のPC側との制御プロトコル

ARCNETが採用されているものとしては，大型印刷機での各機械工程間のネットワーク通信，POSシステムなどがあります．

今回作成したプログラムでも，COM20020Dを使ったARCNETカード(PCI/ISA)を使用することを前提としています．PCで動く自作のプログラムでこのカードを使用する場合，簡単なI/Oポートの記述を行うことで動作させることができます．ISAバス・カードの場合は，ボード上のジャンパでアドレスを指定します．PCI版は，自動的に割り振られるため，UINX上からアドレスを確認する必要があります(後述)．

● ロボット・アームPA10でのARCNETの実装

ロボット・アームPA10では，ARCNETのパケットに独自フォーマットのコマンドを実装しています．ARCNET経由で送られたコマンドをサーボ・ドライバがコマンドを解釈し，モータ制御を行っています．

PA10においてPCとサーボ・ドライバ間でやり取りされるコマンドのプロトコルを図6に示します．図のように，PA10で用意されているコマンドは，S(制御開始)，T(ブレーキ解除)，C(制御中)，E(制御終了)などです．PC側からコマンドを送信すると応答として同じコマンド・データをもつ応答信号がサーボ・ドライバ側から送られてきます．

Cコマンドには，PC側からサーボ・ドライバ側に対して制御入力値データを付けて送信します．この応答データにはPA10の関節角度そのほかのデータもいっしょに送信されてくる構成となっています．

またPA10では，一つのサーボ・ドライバで二つのモータを駆動する構成になっています(図3)．各サーボ・ドライバはARCNETのノードをもっています．したがって，PA10を動作させるには，四つのARCNETのノードと通信する必要があります．旧型のサーボ・ドライバでは，一つのノードに対して2msでの応答が保証されています．最新型のサーボ・ドライバでは，四つのノードに対して1msの制御周期が可能です．

リアルタイム・マルチタスクOS ART-Linuxを採用

● ART-Linuxとは

最初に，ロボット制御のプログラムを作るために，マルチタスクで，かつリアルタイム性を保証したOSが望まれていると書きました．16ビット系のDOSでは(多くの不自由さがあるが)，シングル・タスクなので，あまり難しいことを考えなくても，メカトロニクス制御プログラムを作成することができました．

マルチタスクOSでいちばん使われているのはWindowsです．Windowsは各タスクの切り替えが10msとなっていて，多くのメカトロニクス制御で求められているリアルタイム性より性能が落ちます．また，Windows上ではデバイス・ドライバの作成に関して，かなり詳しい知識が必要となります．また，開発環境も別に購入する必要があるので，敷居が高いところも難点です．

そのほかの商用リアルタイムOSとしては，VxWorksがあります．UNIXと同等の環境も持っているので，ネットワーク系に強みをもっていますが，いかんせん高価です．

昨今，フリーでソース・コードが公開されていること(オープン・ソース)を強みとして，Linuxが台頭してきています．残念ながらLinuxのオリジナル・カーネルのままでは各タスクの切り替えが10msです(カーネル2.4以前)．しかし，ソース・コードが公開されているため，比較的簡単にこのタスク切り替え時間を変更することができます．

また，Linuxのタスク切り替えのタイミングは非常に正確に行われています．オリジナルのカーネルのままでメカトロニクス制御に利用している例もあります．

LinuxにはPOSIXスレッド(pthread)が標準実装されています．Linuxでのスレッドは，ほぼタスクと同等と考えることが

できます．スレッドの場合は，ユーザ・プログラム内のメモリ空間などを共有しています．並列制御のプログラムを作成する場合でも，スレッドを使えばタスク間通信や共有メモリなどの厄介なプログラムを作らずに済むというメリットもあります．

Linuxカーネルを利用して開発されたリアルタイムOSがあります．有名なところではRTLinuxやART-Linuxです．

ART-Linuxは（株）ムービングアイの石綿 陽一氏が開発したリアルタイムOSです．ART-Linuxの特徴としては，以下の点が挙げられます．

▶ユーザ・プログラム内でのリアルタイム制御が可能

▶従来のLinuxアプリケーションやドライバ類がバイナリ互換で使用可能

今回はART-Linux上での実装を行うことにしました．ART-Linuxは現在，カーネル2.2系用と2.4系用のライブラリが配布されています．今回作成したプログラムは，カーネル2.2系用と2.4系用両方のART-Linux上での動作を確認しています．

● ART-Linuxのインストール法（カーネル2.2の場合）

ART-Linuxは（株）ムービングアイの石綿 陽一氏のサイト（http://www.movingeye.co.jp/~you1/art-linux/download.html）からダウンロードできます．

● ART-Linuxのインストール法（カーネル2.4の場合）

まずは，VineLinuxをインストールします．ART-Linuxのカーネル2.4版はVineLinux-2.6のみ対応のようです．

ムービングアイのダウンロード・サービスから，バイナリ・ファイルをメールで申し込みます（http://www.movingeye.co.jp/mi6/download.html）．または，フルパッケージを購入します．以下は，バイナリ・ファイルの場合です．

バイナリ・ファイルを入手した場合，アーキテクチャに対応したバイナリ・ファイルをインストールします．RPMファイルなので，VineLinuxをインストールしたハードディスク上の適当な場所に保存します．root権限で以下のコマンドを実行します．CPUにPentium4を使用している場合は，i686のアーキテクチャになります．

```
# rpm -ihv kernel-headers-2.4.20-0vl29.ART.i386.rpm
# rpm -ihv kernel-2.4.20-0vl29.ART.i686.rpm
```

※Vine2.6r4の場合にはエラーが出るので，強制インストールします．

```
# rpm -ihv --force kernel-2.4.20-0vl29.ART.i686.rpm
```

次に，"/etc/lilo.conf"を編集します．

```
image=/boot/vmlinuz-2.2.16-22_ART20010111
    initrd=/boot/vmlinuz-2.4.20-0vl29.ART
    label=ARTLinux2.4
    read-only
    root=/dev/hda1 # インストール環境により異なる
```

liloの有効化をします．

```
# /sbin/lilo -D
```

再起動して，liloでART-Linuxのカーネルを選択します．cat/proc/versionなどと実行してみることで，ART-Linuxが起動していることを確認できます．

● ART-Linuxによるプログラミング

プログラミングとしては，リアルタイム性をあまり強く意識する部分はありません．制御ループに入るところで，art_enter関数をコールします．引き数に，サンプリング時間と優先度を指定します．ループ内の適当な位置でart_wait関数をコールします．ここで，前回art_wait関数のコールした時刻からサンプリング時間分のスリープを入れた後，プログラムの制御を再開してくれます．

制御ループを抜けたところでart_exit関数をコールします．

```
art_enter();
～～ループ開始～～
  ～制御プログラムを書く
art_wait();

～～ループ抜け～～
art_exit();
```

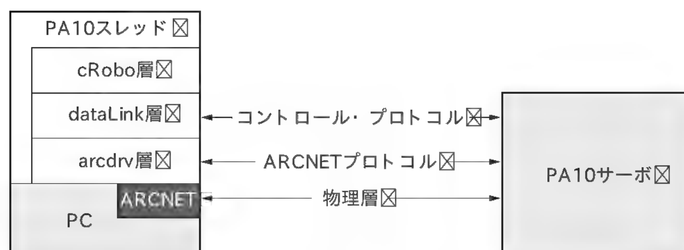


図7　ロボット・アームPA10の制御プロトコル

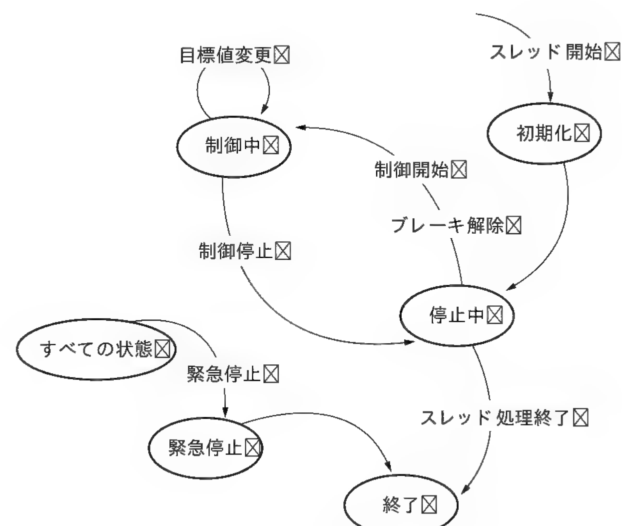


図8　cRobo層の状態遷移図



表3　状態遷移表

		状態				
		初期化	停止中	制御中	緊急停止	終　了
イベント	(遷移時の処理)	Arcnetの 初期化			PA10へ"E" コマンド発行	(無限ループを抜ける) (獲得メモリの開放)
	(通常の処理)	→"停止中"		制御入力演算 PA10へ"C"コマンド発行	→"終了"	
	制御開始		PA10へ"S"コマンド発行 →"制御中"			
	ブレーキ解除		PA10へ"T"コマンド発行 →"制御中"			
	目標値変更			目標値変更		
	制御停止			PA10へ"E"コマンド発行 →"停止中"		
	緊急停止		→"終了"	→"緊急停止"		
	スレッド終了		→"終了"	→"緊急停止"		

ART-Linuxの優れていることの一つとして，あまりサンプリング時間のことを考えずに制御プログラムを組める点があります．もちろん，制御プログラム部分がサンプリング時間に比較してあまりにも処理時間がかかると困ります．

また，もう一つの利点としてユーザ・プログラムとして動作させることができるので，かりに自作プログラム部分にバグがあってもカーネルごと落ちることはないという点もあります．

状態遷移図を使用したプログラムの設計

せっかくマルチタスクで実装するので，ロボット・アームPA10のプログラムを一つのスレッド内(PA10制御スレッド)に記述し，ほかのスレッドでは，PA10以外のデバイスを制御できるように設計しました．

前述のようにPA10では，ARCNETの上に独自のプロトコルを載せて制御を行っています．そこで，PA10との通信プロトコルの階層に合わせて，3層の構造としました(図7)．いちばん下にARCNET通信を制御する層(arcdrv層)．その上に，PA10のサーボ・ドライバを駆動するコマンドをやり取りする層(dataLink層)．さらにその上に，PA10への制御入力演算や各関節角度情報，状態遷移などを行う層(cRobo層)としました．

図8にcRobo層の状態遷移図を示します．cRobo層でもつ状態は，"初期化"，"停止中"，"制御中"，"緊急停止"，"終了"の五つです．また状態間を結ぶ矢印は状態の遷移を表し，太字で書かれたPA10スレッド外部からのコマンド(イベント)により，状態が変化することを示しています．また，状態遷移図を状態遷移表に展開したものを表3に示します．

プログラムの実装

コンパイラにはgccを用いました．ART-Linuxのカーネルに関しては，2.2系と2.4系で動作することを確認しています．

リスト1　cRoboIf.h内で定義している構造体

```
typedef struct
{
    unsigned short intaxisSts;
    double axisDeg;
}cRoboPrm;

typedef struct
{
    double Angle[7];
    void(*callBackFunc)(int _sts,cRoboPrm *data);
}cRoboVal;
```

● cRobo層の実装構成

基本はスレッドの分割で呼ばれる構成をとっています．cRobo層の内部でサンプリングを調整し，PA10への制御および，外部スレッドからのコマンド処理を行います．

以下に，外部スレッドから，PA10制御スレッドを操作する際に使用する(cRoboIf.h内で定義している)構造体を説明します(リスト1)．

"cRoboVal"は目標値設定要求，現在角度要求時に渡す引き数の型です．"cRoboPrm"はコールバック関数で使用する引き数の型です．

cRobo層でPA10スレッド時に呼出される関数"cRoboThread"をリスト2に示します．

cRoboStsFuncTblには，表4に示した状態遷移表に対応する各関数がテーブル化されています．変数"_nowSts"で示されたPA10の状態により呼び出される関数が決定します．

外部スレッドからのコマンドは，キューにぶら下げられて，1サイクルで一つのコマンドが処理されます．ただし，異常終了コマンドだけは，特別にキューの先頭に保持され，優先的に処理するようにしています．

目標値(関節角度)の設定は，やはりコマンドにより行います．また，目標値に到達したときに呼ばれるコールバック関数を指定できます．コールバック関数は，呼び出し側のスレッドで用意する必要があります．

リスト 2　cRobo 層で PA10スレッド時に呼出される関数 cRoboThread

```
void cRoboThread(void *arg)
{
        cRoboSts _nowSts,_bSts;
        unsigned long long tNow,tOld,tDiff;
        int i = 0;

        /* init */
        _nowSts = cRoboSts_Init;
        _bSts = cRoboSts_End;
        newPtr = oldPtr = NULL;

        /* timer start! */
        if(art_enter(ART_PRIO_MAX, ART_TASK_PERIODIC ,
                                          SAMPLING_TIME) == -1){
                printf("artLinux Error!!¥n");
                exit(1);
        }
        callRdtsc(tOld);
        while(_nowSts != cRoboSts_Exit){
                /* 状態遷移時の処理 */
                if(_nowSts != _bSts){
                        (cRoboActFuncTbl[_nowSts])(_bSts);
                        _bSts = _nowSts;
                }

                _nowSts = (cRoboStsFuncTbl[_nowSts])();

                i++;

                /* Wait を入れる */
                if(art_wait() == -1){
                        _nowSts = cRoboSts_End;
                }
                callRdtsc(tNow);
                tDiff = (tNow - tOld) / _CPU_CLK_MSEC_;
                tOld = tNow;
        }
        art_exit();
}
```

表4　状態遷移表(対応関数)

	状　態				
	初期化	停止中	制御中	緊急停止	終　了
遷移時の処理	_cRoboInitAct()	_cRoboStopAct()	_cRoboCtrlAct()	_cRoboEmgrStopAct()	_cRoboEndAct()
通常の処理	_cRoboInitSts()	_cRoboStopSts()	_cRoboCtrlSts()	_cRoboEmgrStopSts()	_cRoboEndSts()

メイン・ループ内で，ART-Linux での処理待機(art_wait)を呼んでいます．ただし，art_wait はただ次のタイマ待ち時間まで待つだけで，処理ループ内の処理が確実にサンプリング周期内で終了すれば，問題ありません．しかし，処理時間がサンプリング周期をオーバしてもそれを知らせてくれることはしてくれないようです．

● dataLink 層の実装構成

前述のように，PA10のサーボ・ドライバと PC 間は ARCNET のパケット上に独自のデータ・フォーマットのコマンドを送受信することで成り立っています．この部分を実装しているのが dataLink 層です．

リスト 3に，送信データを定義した構造体の形式を示します．

ここで，ctrlSts は制御用データです．また tlkInp は入力トルク値(トルク制御時)，speedInp は入力速度値(速度制御時)を示しています．速度制御かトルク制御かは，ctrlSw のビットにより決定します．

リスト 3　送信データを定義した構造体の形式

```
/* Send Data format PA10 */
typedef struct{
    union {
        unsigned char Byte;
        struct {
            unsigned char blkSw:1;
            unsigned char servoSw:1;
            unsigned char ctrlSw:1;
            unsigned char deadSw:1;

            unsigned char bit04:1;
            unsigned char bit05:1;
            unsigned char bit06:1;
            unsigned char bit07:1;
        }Bit;
    } ctrlSts;
    unsigned short int tlkInp __attribute__((packed));
    unsigned short int speedInp  __attribute__((packed));
} ctrlData;
```

また，__attribute__((packed))は，構造体のアライメントをなくすオプション(アライメント・パック)を指定しています．アライメント・パックを指定しない場合，コンパイラは CPU のアクセスしやすいメモリ番地に変数領域を確保します．たとえば 32ビット処理系の場合は，各変数を 4の倍数のアドレスに配置していきます．

リスト 4に受信データを定義した構造体の形式を示します．

リスト 4で，axisSts はロボットの状態情報が入っています．axisRez は各関節の角度情報が入っています．axisTlk は，各関節モータのトルク値を示しています．

● I/O ポートの使い方

Linux では，ユーザ・プログラムを実行するとカーネルよりメモリ空間が割り当てられます．しかし，ユーザ・プログラムから I/O ポートを直接アクセスすることはできません．

しかし，iopl 関数に引き数“3”を渡してコールすると，ROOT 権限で実行されたユーザ・プログラムから I/O ポートを駆動させることができるようになります．

また，I/O ポートにバイト・データを入力する場合には inb(port)を呼び出します．逆に，バイト・データを書き出すには outb(value,port)を呼び出します．outb は，DOS などの処理系と引き数の順番が逆になっているので注意が必要です．

前述のように PCI バスの I/O ポート・アドレスは，自動的に割り当てられます．このため，Linux 上からポート・アドレス



リスト 4　受信データを定義した構造体の形式

```
/* Recive Data format PA10 */
typedef union {
    unsigned short int Word;
    struct{
        unsigned short int axLmtA:1;
        unsigned short int axLmtN:1;
        unsigned short int bit02:1;
        unsigned short int bit03:1;

        unsigned short int bit04:1;
        unsigned short int abortSw:1;
        unsigned short int ctrlMode:1;
        unsigned short int refSigSts:1;

        unsigned short int blkSigSts:1;
        unsigned short int ipmErr:1;
        unsigned short int tlkErr:1;
        unsigned short int axLmtOver:1;

        unsigned short int refErr:1;
        unsigned short int speedErr:1;
        unsigned short int comErr:1;
        unsigned short int servoSts:1;
    }Bit;
} servoSts;

typedef struct {
    servoSts  axisSts;
    long int  axisRez;
    short int axisTlk;
} axisData;
```

をチェックし，プログラムでの設定も書き換える必要があります．以下のコマンドでARCNETカードのI/Oアドレスを確認できます．

```
# cat /proc/pci
```

実装と実験結果

図9は，PA10動作中のサンプリング時間のずれを示したグラフです．グラフの横軸が，動作プログラムの制御回数を示しています．また縦軸が制御時の実際に計測したサンプリング時間を示しています．

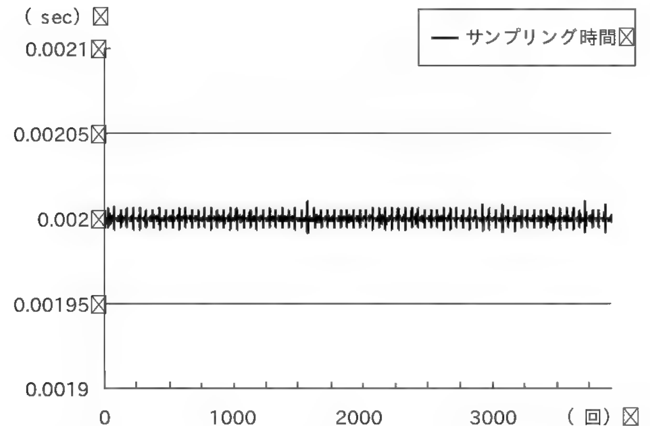


図9　ロボット・アームPA10動作スレッドのサンプリング周期の変動

この実験の際，PA10スレッドの優先度はいちばん高くして実行しています．また別のスレッドで画像処理ボードを動作させています．しかしその影響を受けることなく，サンプリング周期の変動はほぼ± 1%以内で収まっていることがわかると思います．

● プログラムのダウンロード先

今回開発したPA10動作プログラムは，以下のサイトよりダウンロード可能です．

http://www.egamilab.org/artlinux4pa10/download/download.cgi

よしかわ・ともやす　フリー・プログラマ

組み込みプログラミング・ノウハウ入門(第20回)

文章からのクラス抽出

――日本語で書かれた仕様書をUMLへ変換する

藤倉 俊幸

前回までは組み込みシステムにおけるスケジューリングの話を取り上げてきた．今回はスケジューリングから離れ，組み込み設計におけるクラス設計について述べる．

オブジェクト指向において，クラスを設計することは難しいという声がある．そこで日本語で書かれた仕様書からクラスを抽出する手法を紹介しよう．

1 クラス抽出は難しいか

オブジェクト指向を始めて最初にぶつかる壁は，何をクラスにしたら良いのかがわからないことである．しかし一方で,「オブジェクト指向では"もの"を単位にしてモデリングするので，自然に開発対象物をモデル化しやすい」とも言われる．この"もの"がクラスに対応するはずである．であれば，クラス抽出は簡単でなければならない．

"もの"にもいろいろある――物理的なものから，抽象的なもの，形式的なものなど．ところが，物理的なクラス以外を抽出することがとくに難しいと言われている．たとえば,「クラス抽出とは難しい"もの"だ」という文章の"もの"は，難しい部類に入る．どんなものでも，ものともせずにモデル化してしまう方法はないものだろうか．

そこで，自然言語で記述された仕様書と，プログラミング言語の世界の中間に属するUMLをもう少し積極的に利用することで，クラス抽出を容易にする方法について提案する(図1)．

今年に入ってから概念モデルを作るのに役に立つ本が出版されている[1][2]．ここで述べる手法は，概念モデル構築に入る前のたたき台をつくる際に使うことができる．組み込みシステムを作る際の全体フローの中での位置づけを図2に示す．実際に使用する際には，文章は個人によってばらつきが大きいことを考慮する必要がある．文章からたたき台を作って，その中から本質的なものを切り出して概念モデルとしなければならない．

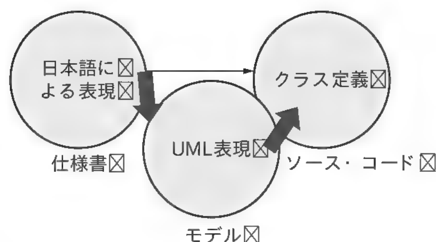


図1 UMLを経由する

2 例題1～電子ポットの温度制御～

例として参考文献(3)のユースケースからクラスを抽出してみる．題材は「電子ポットの温度制御」でユースケースは図3のようになっている．ここでは原文のまま引用する．この例題に関する詳細は参考文献(3)とその元になっているSESSAMEの「話題沸騰ポット(GOMA-1015型)要求仕様書」[4]で参照できる．

● 基本系列1)

図3の基本系列1)の部分のみをモデル化すると図4のようになる.「ポットの沸騰ボタン」という記述があったので，沸騰ボタンはポットの部分とした．この文章の文型である「AのB」を利用する．そして，アクタである「お茶を飲みたい人」が「押下した」は，文型「AがBをCする」を利用する．

ここで使用した導出ルールは，

1) 名詞をオブジェクトにする
2) 他動詞を関連にする
3) 所有関係は集約にする

の三つである．

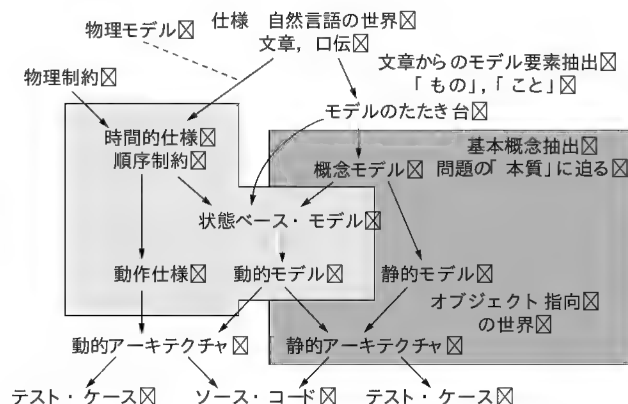


図2 全体フロー

▶ ユースケース名：水を沸騰させる
▶ 概要：ポットの中に入っている水を沸騰するまで加熱する
▶ アクタ：お茶を飲みたい人
▶ 事前条件：ポットに水が入っていること，ポットのふたが閉まっていること
▶ 事後条件：ポットの水が沸騰していること
▶ 基本系列：
1）アクタは，ポットの沸騰ボタンを押下する．
2）システムは，温度制御可能な水位かどうかをチェックし，水位が不十分であればこのユースケースを終了する．
3）システムは，沸騰ランプを点灯する．
4）システムは，ヒータの操作量の算出を行い，ヒータで加熱する．
5）システムは，サーミスタをチェックし，現在の温度を表示する．
6）システムは，加熱時のエラー検知アルゴリズムにより，エラー検知を行う．
7）システムは，現在温度が100℃に到達するまで，4）〜6）を繰り返す．
8）システムは，現在温度が100℃に到達したら，3分間加熱を続ける．
9）システムは，3分間加熱を行ったら，沸騰ランプを消してブザーを鳴らす．
10）アクタは，ブザーを聞いて，水が沸騰したことを知る．

図3　電子ポットの温度制御のユースケース

● 基本系列2）

次に基本系列2)をモデル化してみる（図5）．文章としては「システムは，ユースケースを終了する」が骨格であると思われる．クラス抽出が目的なので，静的な構造を記述している残りの部分にまず注目する．すると，「システムは，水位をチェックする」，「水位が不十分である」が単文として抽出されるので，これをモデリングする．文型「AはBをCする」と「AがBである」を利用する．

この文章では「温度制御可能かどうか」が「チェックする」にかかるので，関連クラスを導入して表現した．関連クラスは，多対多の関係を実装するために導入されるが，オブジェクト抽出の段階では抽象的なオブジェクトをとらえることに利用できる．抽象的なオブジェクトの抽出は，初心者には難しいと言われているが，関連クラスを利用すると比較的簡単に抽出できる．関連は動詞から導出されて，関連クラスは関連をベースにして導出されるので，動詞から名詞を作ることで抽象的なオブジェクトを見つけ出すことができる．また，形容詞からも抽象的なオブジェクトを導出できる．

A）動詞から，読む→読み方/読み手，歩く→歩き方
B）形容詞から，早い→早さ，強い→強さ

「水位が不十分である」の部分は，水位の状態について述べているので「水位不十分」という状態を水位クラスに追加した．そして，「ユースケースを終了する」の部分は，この水位不十分状態のエントリ・アクションとした．文型「AがBのときCする」を利用した．

ここで使用した導出ルールは，

4) 動詞に副詞句などが掛かっていたら，関連クラスを導出する

である．

● 基本系列4）

単純な基本系列3)は飛ばして，基本系列4)をモデル化してみる．「システムは，ヒータの操作量の算出を行い」の部分は

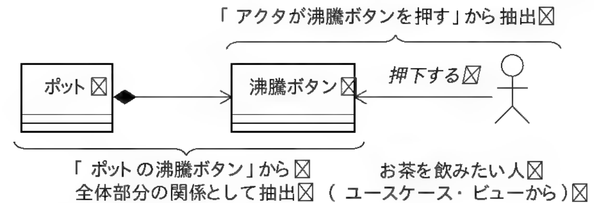


図4　基本系列1)のモデル化

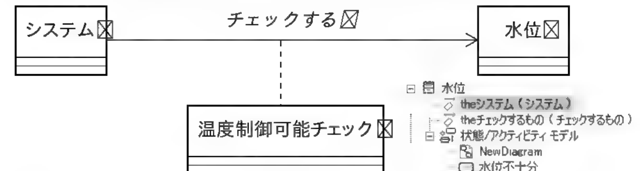


図5　基本系列2)のモデル化

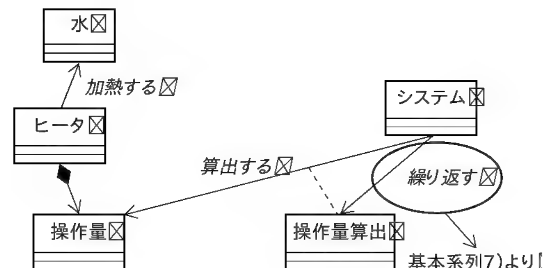


図6　基本系列4)のモデル化

「システムが，ヒータの操作量を算出する」と読み換えた．また「ヒータで加熱する」は「ヒータが水を加熱する」と解釈した．関連の「算出する」は，基本系列の7)で「繰り返す」という記述があるので関連クラスを導出した．

図6で新たに使用したルールは，

5) 動詞に対する動詞がある場合は，先の動詞に対する関連から関連クラスを導出する

である．

● 結果の評価

以上，6種類の導出ルールを使って，ユースケース分析を行うと全体のクラス図として以下のものを得ることができる（図7，p.208）．この程度のクラス図であればユースケース記述を読みながら10分程度で作ることができる．しかも，文型パターンと導出ルールを用いることで，ルーチン・ワーク的にクラス図を作ることができるので，初心者でも抽象概念を用いたモデリングが可能になる．

オブジェクト指向初心者の場合，たとえば参考文献（3）によれば，図8のように，ランプ，ポット，ヒータ，センサのような具体的な“もの”のみを抽出してしまう．図7のような，水，水位，温度，操作量，温度制御などの抽象的なクラスは出てこない．これらの抽象的なものをクラスとしてとらえられれば，

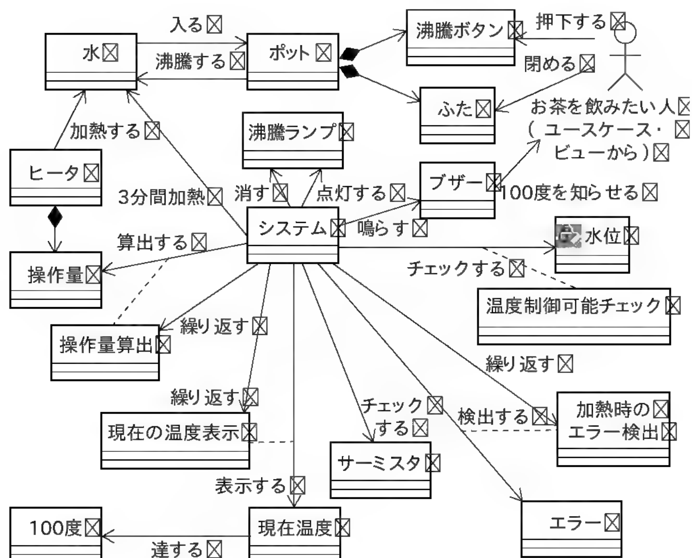


図7　導出されたクラス図



図8
初心者のクラス図（文献1より）

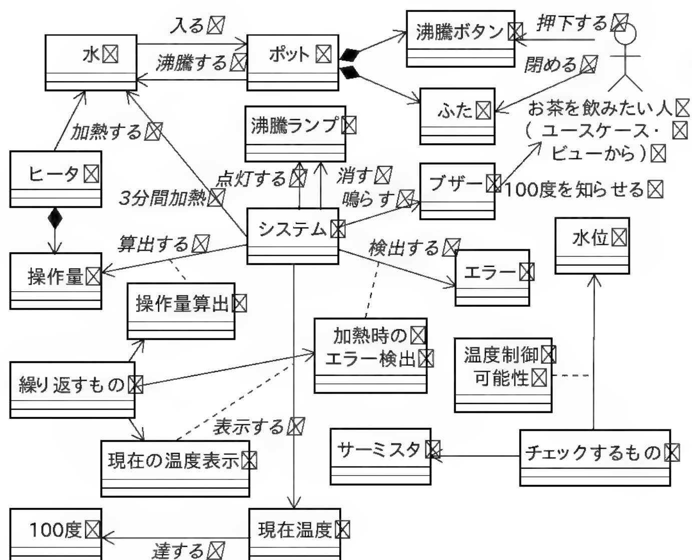


図9　クラス・システムを一部分割したクラス図

リファクタリングを実施することで，より良いシステムを作っていける．ランプ，ポット，ヒータ，センサのような具体的な物だけでは，たとえばポット・クラスあたりがだんだん肥大化してしまい，オブジェクト指向を使わない場合と同じことになってしまうことが多い．

文章からのクラス抽出で重要なのは，

「文章に書いてあることは100%UMLモデルに移行する」

という努力である．変なクラス図になっても気にしないことだ．100%移行できれば，ビジュアル的に考察するだけになるので，文章を読むよりも効率が良くなる．皆でディスカッションするにはちょうど良い．

一応は完成したクラス図だが，もう少しブラッシュアップしてみよう．図7では，まず中央にいる「システム」が目障りである．これでは，ポット・クラスが肥大化するように，システム・クラスが開発の過程で肥大化してしまう．原因は，ユースケース記述に「システムは，…」という文章が多いためである．日本語では，「は」は必ずしも主語を表すわけではないが「システム」を主語と解釈して分析した結果である．これを防ぐためには，ユースケースを記述する際に，主語と目的語はなるべく正確に表現したほうが良い．

モデリング的には中央の「システム」を取り除きたい．そのためには，「繰り返す」，「表示する」，「チェックする」などの動詞の主語を仕様書から探せばよい．そのような情報源がない場合は，「繰り返す」と「チェックする」は複数回現れているので「繰り返すもの」，「チェックするもの」というクラスを導出して「システム」クラスを分割する（図9）．分割後，クラス図の変化を見て適当な名前に付け替えればよい．ただし，安易にやりすぎると手続き的な構造を導入してしまうことになる．しかし，その一方で割り込みハンドラなど裏方の構造を表に引っ張り出すことができる．

3 例題2～化学プラント警報システム～

次に，例として参考文献（5）の「化学プラント警報システム」からクラスを抽出してみる．参考文献（5）は，VDM（Vienna Development Method）と言う形式仕様記述ツールに関する解説書である．ここで，利用する例題の簡単な解説はWebサイト[(6)]からもダウンロードすることができる．形式仕様記述についてはあまりなじみのない概念だと思うので，ぜひダウンロードして読むことをお勧めする．形式仕様記述を使えば，参考文献（6）のまとめで言われている「非形式文書はあいまいで未完成で矛盾している」というその矛盾を，要求の段階で見つけてしまうことができるのである．では，さっそくその矛盾している要求仕様を見て見よう．

参考文献（5）と（6）の文章は，同じ内容を表しているハズであるが微妙に違っている．ここでは，参考文献（5）から原文のまま引用する．

R1：このプラントの警報を管理する計算機システムを開発する．
R2：警報に対処するため，4 種類の専門分野が必要である．すなわち，電気・機械・生物・化学の4 種類である．
R3：システムが稼働している期間では常に，専門家が勤務していなければならない．
R4：各専門家は各自の専門分野の一覧を持つことができる．
R5：システムに報告される警報には，おのおの専門分野が指定されており，警報には専門家が理解できる記載事項が付いている．
R6：システムが警報を受け取ると常に，適切な専門分野の専門家を見つけて呼び出さなくてはならない．
R7：専門家は，システムのデータベースを用いて，自分たちがいつ勤務するのかを確認できる．
R8：勤務している専門家の数を知ることができなければならない．

図10　化学プラントの概要

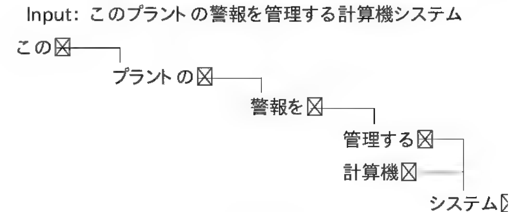


図11　日本語構文解析システムの結果

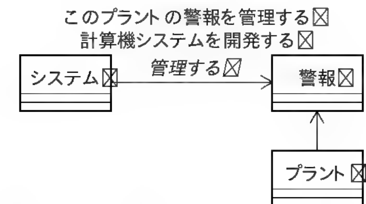


図12　要求1のモデル化
このプラントの警報を管理する計算機システムを開発する

● 化学プラントの要求

化学プラントには，プラント内の条件に反応して警報を発生することができる多数のセンサが装備されている．警報が発生すると，専門家を現場に呼び出さねばならない．各専門家は，異なる種の警報に対処するために異なる専門分野を担当している．このシステムの概要を図10に示す．

● モデリングの目的と位置づけ

プログラムを書く場合は，それをコンピュータで動かすことが目的なので，目的がわかりやすいし，見失うこともない．しかし，モデルという抽象的なものの場合，肝心の目的があいまいになってしまう場合がある．とくに，MDA など動くモデルの場合は，動かすことに注意が向いてしまい，モデル本来の目的を忘れてしまうことがある．MDA をビジュアル・プログラミング環境と割り切った人にはどうでも良いかもしれないが，やはりモデリングは仕様を検討したり，アーキテクチャを検討するために利用するのが本来の目的である．一方で動かないモデルを作っている場合は，科学者になったり哲学者になってしまうことがある．

参考文献(5)におけるこの例題本来の目的は，要求を実現するために必要な規則を明確にすることである．どのような規則かと言うと，情報の管理方法や，専門家を呼び出す規則など，プログラムを実現するために必要な規則である．これらの規則が，プログラム仕様となる．あるいは，その規則に従ってユースケースのイベント・フローを記述する．

一方，この記事で例題2に対して行おうとしていることは，「仕様を記述するために，必要なデータ構造とその間の関係を文章のみを手がかりとして機械的に抽出すること」である．自然言語で記述された内容をデータ構造に写し取ることは，非形式的な記述を形式的な記述に変換する第一歩である．第二歩目は，抽出したデータ構造あるいはクラスに不変条件などを追加することになる．ここでは，第二歩目は対象としていない．

● 要求を読む

▶ 要求1

「開発する」は除外する．すると残りは，

　このプラントの警報を管理する計算機システム

となる．この名詞節を文章にすれば，

　A：計算機システムがこのプラントの警報を管理する
　B：この計算機システムがプラントの警報を管理する

となる．「この」は，「プラント」にかかると考えるのが一般的である．つまりA のほうが一般的である．ここでは，「この」自身は重要ではないと思われるので省略する．

省略できない重要な単語の場合には，かかり受けの多様さはあいまいさの原因になる場合が多い(後で紹介するツールの項目を参照)．一般的なかかり受けを知りたい場合には，文章データを持っているツールを利用すると良い．たとえば，この例文を京都大学が公開している構文解析ツールにかけると図11のような結果が得られる．

要求1はA を使って，

　計算機システムがプラントの警報を管理する

となる．

「プラントの警報」という記述に対して「A のB」文型が使えるか検討する．この場合は，

　プラントが発した警報
　プラントからの警報
　プラントに関する警報

などで置き換えることができるので，全体/部分の関係ではないと考えられる．ただし，「プラント」も「警報」も，どちらも重要そうなので，単純な依存関係を使用して「プラント」と「警報」の間の関係を残すことにする．この関係は将来見直される可能性が高い．

要求1のモデル化は「A がB をCする」文型を使用して図12のようになる．

▶ 要求2

要求2は，構文解析ツールを使用して構文解析すると二つの部分に分割される(図13)．

　A：「何か」が，警報に対処する

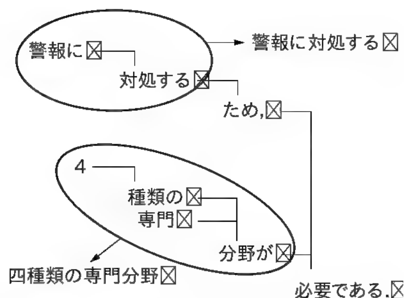


図13　要求2の構文

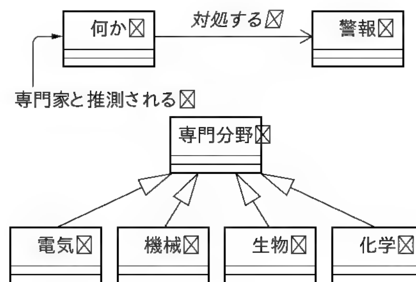


図14　要求2のモデル化

警報に対処するため，4種類の専門分野が必要である．すなわち，電気・機械・生物・化学の4種類である

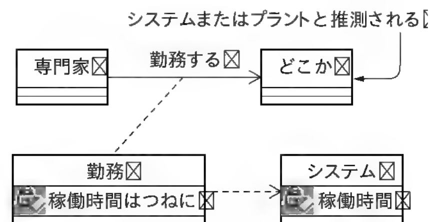


図15　要求3のモデル化

システムが稼働している期間では常に，専門家が勤務していなければならない

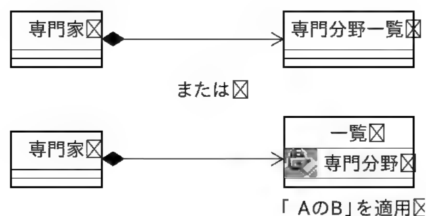


図16　要求4のモデル化

各専門家は各自の専門分野の一覧を持つことができる

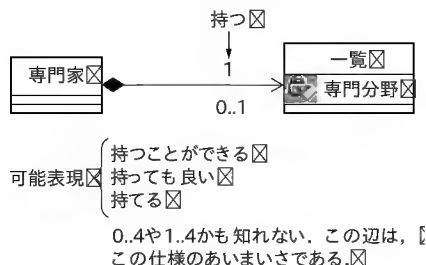


図17　持つことができる

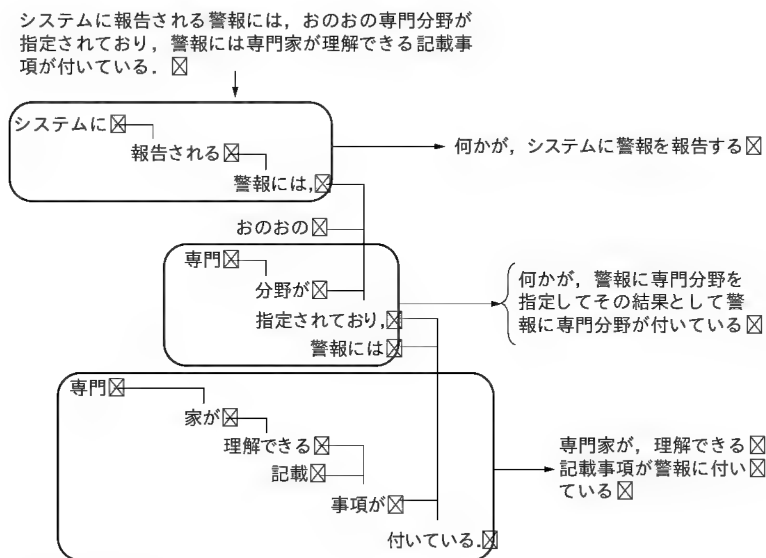


図18　要求5の分解

B：四種類の専門分野(電気，機械，生物，化学)がある

の二つに要約される．

「何か」は「対処する」の主語がわからない状態で「AがBをCする」文型を使用するために導入した．これは，将来，主語が明確になった段階で置き換わることになる．要求の概要に,「各専門家は，異なる種の警報に対処する…」とあるので「何か」は，専門家であることがわかる．さらに,「各…は，異なる…」から専門家と警報の間には多対多の関係が予想される.「…種類のA」は，aKind-of関係なので継承関係を導入する(図14).

ここで新たに使用した導出ルールは，

6) 文型適用時に対応するものがないときは,「何か」,「だれか」,「どこか」などを補う

である．この導出ルールは仕様の漏れを検出するプローブとして利用できる．

▶ 要求3

「システムが稼動している期間」から,「システム」は「稼動」状態を持っている.「専門家が勤務して…」には何処に勤務するか記述されていない.「システム」または「プラント」だと思われるが，とりあえず「どこか」とすることで「AがBにCする」文型を使用できるようになる．さらに，どのように勤務するかについての記述があるので関連クラスを導出する．基本的には，図15を使用するが,「稼動状態」がシステム・クラスの状態なので導入した関連クラスからシステム・クラスに依存線を引いてモデル化した．

▶ 要求4

これは，単純に所有関係としてモデル化する(図16).「各自の」となっているので集約ではなくコンポジションとする.「…できる」あるいは「…ことができる」は，能力があること，あることが可能な状況にあることを表現する(図17)．仕様書では「能力の可能」と「実現」とどちらなのかが問題になるあいまいな表現につながる.「可能」を表現したければ，一覧クラスの多重度を0..1とすることで表現できる．

▶ 要求5

要求5の文章は長いので，図18のように三つの部分に分解してモデリングする．難しいのは「されており」の部分の解釈で,「する」という動作に注目するか，動作が完了した状態に注

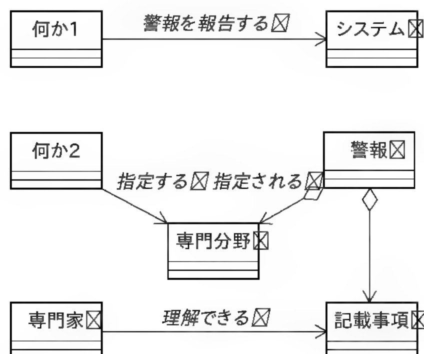


図19　要求5のモデル化

システムに報告される警報には，おのおの専門分野が指定されており，警報には専門家が理解できる記載事項が付いている

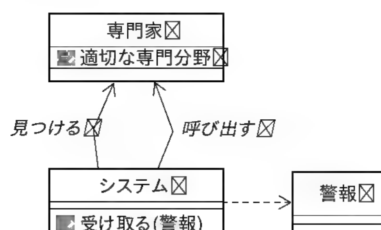


図20　要求6のモデル化

システムが警報を受け取ると常に，適切な専門分野の専門家を見つけて呼び出さなくてはならない

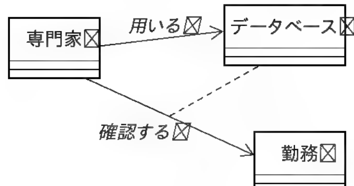


図21　要求7のモデル化1

専門家は，システムのデータベースを用いて，自分たちがいつ勤務するのかを確認できる

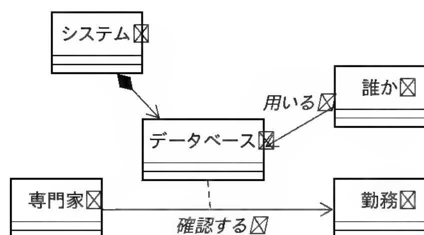


図22　要求7のモデル化2

専門家は，システムのデータベースを用いて，自分たちがいつ勤務するのかを確認できる

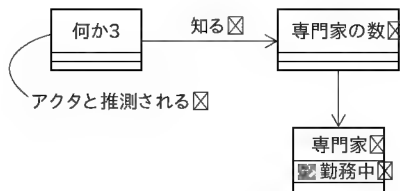


図23　要求8のモデル化

勤務している専門家の数を知ることができなければならない

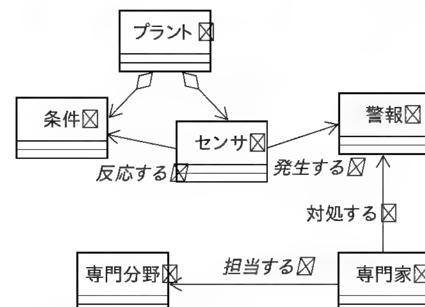


図24　概要のモデル化

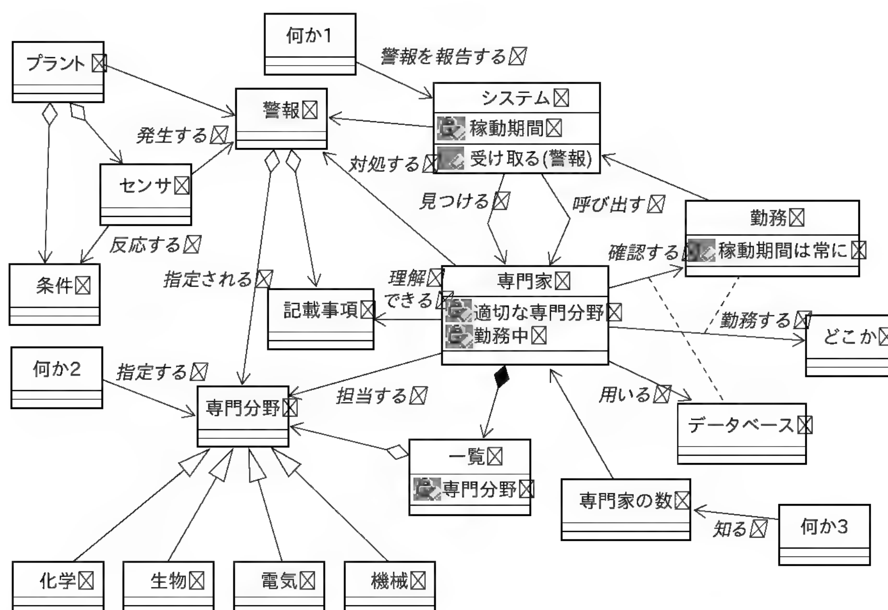


図25　全体のモデル

目するかによって，図19のように2通りにモデリングできる．この場合，動作に注目すると主語が必要になり，状態にも注目すると警報に専門分野が付属している形になる．完了形の場合には状態が重要な場合が多い．

▶ 要求6

「システムが警報を受け取る」の部分はすでに，要求5の「何かが，システムに警報を報告する」でモデル化してある(図20)．

▶ 要求7

システムのデータベースを用いるのが専門家であれば図21，かりに仕様書に記述されていないだれかであれば図22のようにモデル化できる．

▶ 要求8

だれが知りたいのかがはっきりとしない．ユースケースを使用していれば恐らくアクタが登場する場面だと思われる(図23)．

▶ 全体

概要の部分のモデリングを図24に示す．今までのモデリング結果を一つのクラス図に表すと図25になる．

「何か」が三つと「どこか」が一つ残っている．専門家が勤務する「どこか」は多分，プラントだろう．警報の専門分野を指定する「何か2」は人なのか，エキスパート・システムなのか不明である．「何か1」と「何か3」もよくわからない．まず，これをインタビューなどによってはっきりさせる必要がある．

次に，クラス図上に多重度を入れて数の関係を決めていく．たとえば，概要に「多数のセンサ」と記述されていることから，センサは複数であることがわかる．数の関係を文章から抽出できない場合には，やはりインタビューなどによって調べる必要がある．参考文献(5)では，警報に指定される専門分野が一つだけなのか，少なくとも一つだけなのかがあいまいであると指摘されている．クラス図に数の関係を入れることで，このへんの関係を確認することができる．参考文献(5)では，分析の過程で「日程表」の必要性を認めている．ここで作成したクラス図では，勤務クラスによって日程表に対応する役割が抽出されている．

4 使用したルールと文型パターン

● 導出ルール

使用した導出ルールを図26にまとめて再掲する．

● 文型パターン

▶ AのB

「の」にはいろいろな意味があるが，所有や所属，所在を表している場合(has_a関係)に図27のパターンを利用する．ただし以下の場合には適用しない．

1) 性質・属性を表す場合，「病気の人」，「バラの花」，「3時の電車」，「ヒゲの男」．AがBを説明する．AはBの属性または属性値になる場合もある．
2) 同格の場合，「友人の和男」，「50周年記念品の江戸切子」など．この「の」は「である」で置き換えられる．AとBは同じものを指している．is_a関係(クラスとそのインスタンス)になる場合もある．
3)「が」や「を」で置き換えられる場合，「自転車の修理」，「子供の成長」，「彼の描いた絵」，「花の咲くころ」
4) 行為者関係，「偉人の業績」，「遺族の悲しみ」

上記の3)や4)の場合は，図28のように暫定的にAからBに依存関係を引いておく．

▶ AがBに(を)Cする

Aが主語で，Bが目的語で，Cが動詞の場合．いわゆる他動詞構文の基本文の場合である．「が」は「は」になることもある．日本語の場合，同一の内容を基本文，受身文，自動詞文で表現することができる．

基本文： AがBをCする．「警官が泥棒を捕まえた」
受身文： BがAにCされる．「泥棒が警官に捕まえられた」
自動詞文： BがAにC'する．「泥棒が警官に捕まった」

日本語では自動詞文が多用される傾向にあるので，どちらが主語になるのか混乱する場合もある．仕様書のような文語体では自動詞文はそれほど多くないが，口語体では自動詞文が多用される．普通，仕様書を読んで頭の中で考えているときは口語体を使用するので油断できない．英文の仕様書を読んでいても同様である．仕様書でよく出てくる「行う」，「送る」，「読む」などには自動詞文がないが，そのほかの動詞については注意する．関連名には「捕まえる」のような基本形を使用するので，自動詞文のときは基本文に直してからUML化する．「AがBにCする」も同様に扱う(図29)．

動作に対して条件が付いている場合には，動詞を名詞化したまたは副詞句を名詞化した関連クラスを導入して表現する(図30)．

● 動詞文と名詞文

▶ AがBにCをDする

英語の第四文型に対応させるとBが間接目的語でCが直接目的語になる(図31)．最終的に実装に落とす場合，Dというオペレーションに Cというパラメータを渡す場合が多いので，AからBに「CをDする」という関連を引く．

「Dされている」と完了形になって結果の状態が重要な場合には図32のように分割して表現する．

▶ Aがbである

Aの状態を断定または定義する．「が」は「は」になることもある(図33)．

bが名詞・形容詞の場合は文末が「だ」，「である」，「です」に

1）名詞をオブジェクトにする
2）他動詞を関連にする
3）所有関係は集約にする
4）動詞に副詞句などがかかっていたら，関連クラスを導出する
5）動詞に対する動詞がある場合は，先の動詞に対する関連から関連クラスを導出する
6）文型適用時に対応するものがないときは，「何か」，「だれか」，「どこか」などを補う

図26 導出ルール

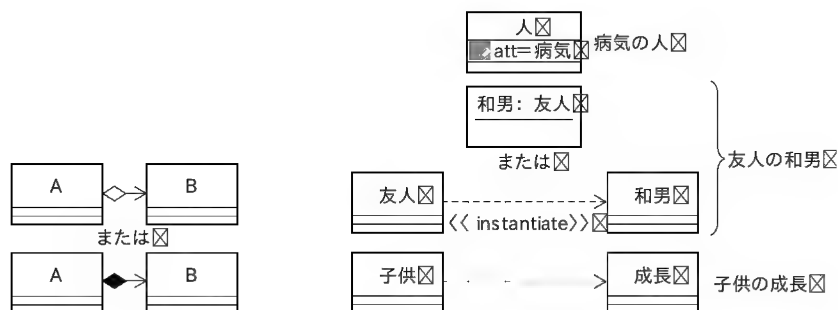


図27 AのB所有の場合

図28 AのB所有でない場合

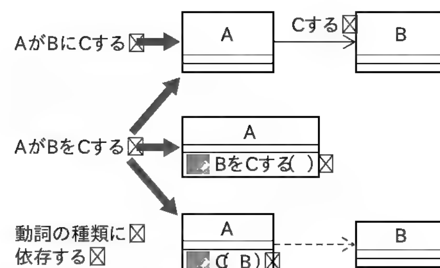


図29 AがBに(を)Cする

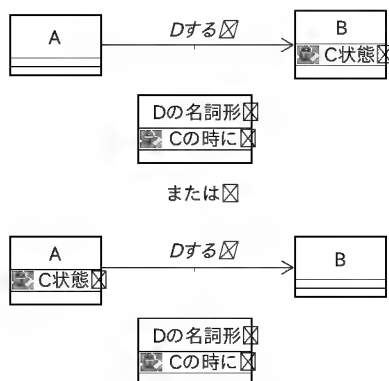


図30　AがBにCのときDする

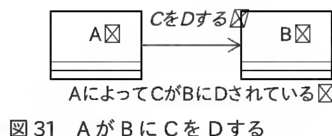


図31　AがBにCをDする

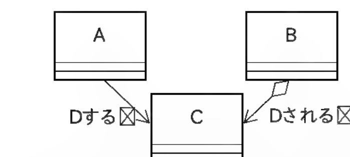


図32　AによってCがBにDされている

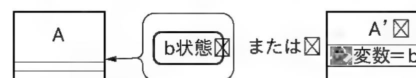


図33　Aがbである

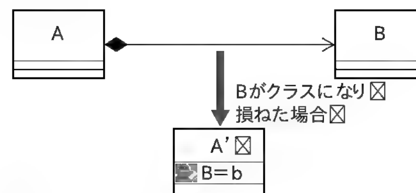


図34「AのB」と「Aがbである」

なる．例:「今日は雪だ」

bが動詞・形容詞の場合は言い切り型になっている場合もある．例:「頭が痛い」,「雨が降る」

文型「AのB」で抽出したBが最終的にクラスにならずにintなどの基本形になって「Aがbである」のbを格納する属性になる場合もある(図34).

▶ AがBのときCする

「が」が「は」になることもある(図35).

▶ AがBをCしてDをEする

図36のようになる．

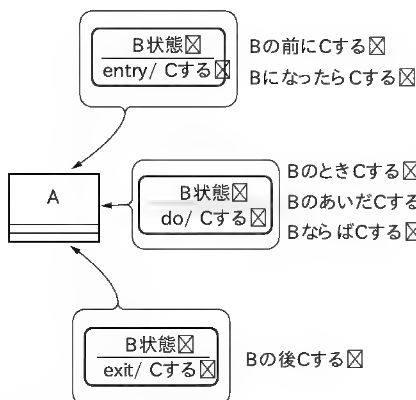


図35　AがBのときCする

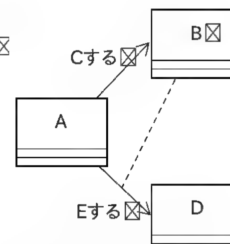


図36　AがBをCしてDをEする

5 文書について

例題1では，ユースケース文書を対象にしてクラス抽出を試みた．ユースケース文書は，イベントのシーケンスが記述してあるので，比較的，主語や述語が明確になっており，ここで述べた手法を適用しやすい．また，システム化の範囲が確定している場合が多いので，この手法を使用してもシステム化の範囲やシステム化の目的を拡大してしまう危険性は低い．企画書とかシステム構想書などにこの手法を適用する場合には，どんどん話が大きくなって壮大なクラス図ができあがってしまうことがある．

この連載の意図は，なるべく個人の能力に依存せずにだれでもある程度のレベルでモデリングできる方法を探ることである．なので，抽出の過程で常識やドメイン知識などの個人の知識や能力に依存するものを利用せずに分析を進めている．その結果，文章だけがよりどころとなり，文章表現の揺らぎの影響を受けやすい．しかし，あえてそのままモデリングすることで文章を形にして見ることができる．そうすることで，グループで議論をするベースを作ることができる．また，どのような文章がモデリングしやすいのかがわかってくるのではないだろうか．

例題2は，例題1に比べると文章は複雑である．構文解析の力を借りて単文にしてからUMLに変換した.「できる」,「ねばならない」,「つねに」,「ている」などUMLに変換できない表現がある．

6 ツールについて

● UMLツール

この手法を適用するためには，日本語をスパスパ入れられるツールがあれば非常に便利である．いちいち，仕様ビューなどをオープンしないと日本語を入れられないツールでは，リズムに乗れない．先ほどの図7を10分で完成させるには必須である．

最近はフリーなUMLツールも利用できるようになったが，関連クラスや状態を定義できないものなどもある．少なくとも関連クラスと状態を入れられないと使えない．

要求ごとに作成したクラス図は図37のように保存しておくことで，要求が変化したときに追跡することが容易になる．要求ごとにクラス図を描いていって，抽出したクラスを一つのクラス図上にドラッグすれば全体のクラス図が勝手にできあがるUMLツールでなければ，この記事で説明したことは使えない．そのため，単なるお絵かきツールではあまり意味がない．全体のクラス図を見ながら，構造を変化させた結果が各要求のクラス図にどのような影響を与えるか確認したり，個別の要求の解釈を変更した際に全体のクラス図にどのような影響があるか確認できる必要がある．

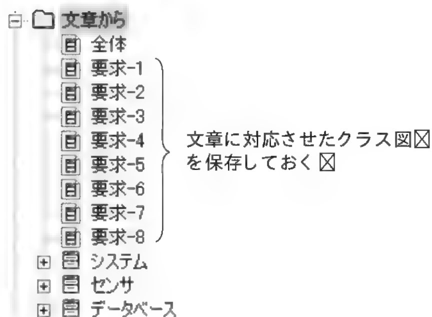


図37 要求の管理

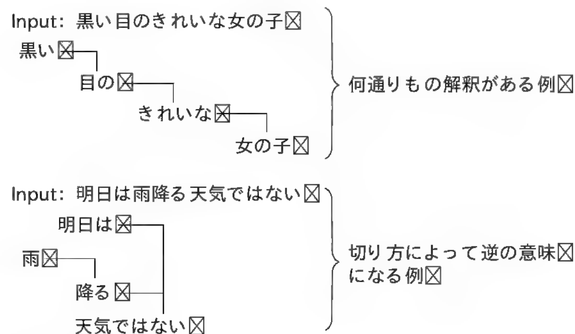


図38 KNPの結果

● 構文解析ツール

一般的に受け入られていて，高い確率でなりたつ日本語のかかり受け関係には以下のものがある．これらの規則は，書きことばでは話しことばよりも高い確率で成立する．

1) 文節はもっとも近い文節にかかる
2) かかり受け関係は交差しない
3) 優先度の高いものが低いものにかかる

実際のアルゴリズムについては自然言語処理の教科書に詳しい．また，各種構文解析ツールを利用すると良い．開発部門独自のコーパス(corpus)[注1]を構築することで，解析精度を向上させることができる．

京都大学では，これらの規則を実装したプログラム「KNP」を公開[7]している．判断に困ったときは，KNPを利用してどの単語がどの単語にかかるのが一般的なのかを調べることができる．ちなみに，有名な例文「黒い目のきれいな女の子」と「明日は雨降る天気ではない」をKNPにかけると図38のような結果が得られる．

一般的なかかり受けで納得できない場合は，可能なかかり受けを列挙する，Rubyで書かれたパブリック・ドメインのツール「日本語文解析プログラム」[8]もある．このツールは，コーパスを持っていないので「黒い目のきれいな女の子」に対して「目の子」のようなかかり受けも出力される．

おわりに

明治の初めのころまで，話しことばと書きことばが完全に分離していて，それぞれ考えたり，議論するための道具ではなかった．話しことばは日常の猥雑(わいざつ)なものに徹して，書きことばは型にはまった漢文訓読体だった[9]．書きことばは，「訳がわからないほどありがたがられる」などという構造ができていたという．話しことばが動くプログラムに対応して，書きことばが仕様書とモデルに対応しないだろうか．コードを見ないと考えられないというのは，仕様書とモデルと実装コードが乖離していて，それぞれがまともな考えるための道具になっていないのではないだろうか．

注1：大量に収集された言語データ．辞書，新聞，雑誌，小説，論文などの文章を集めて構文解析し意味情報などを付加したもの．

参考文献・URL
(1) 児玉 公信；UMLモデリングの本質，日経BP，2004.
(2) 渡辺 博之，芳村 美紀，桑本 茂樹，敷山 喜与彦；思考系UMLモデリングエクササイズ，翔泳社，2004.
(3) 山田 大介，久我 雅人；「できる技術者」のクラス抽出はこんなに違う!，Design Wave Magazine，2003 Feb，pp. 114 - 121. http://www.cqpub.co.jp/dwm/Contents/0063/dwm006301140.pdf
(4) 組込みソフトウェア管理者・技術者育成研究会(SESSAME)，http://www.sessame.jp/
(5) ジョン・フィッツジェラルド，ピーター・ゴーム・ラーセン著；荒木 啓二郎，張 漢明，荻野 隆彦，佐原 伸，染谷 誠 訳；ソフトウェア開発のモデル化技法，岩波書店，2003，ISBN4-00-005609-3 C3004
(6) http://shinsahara.com/www/vdm/index.html の「日本語版VDM++入門コース」をダウンロードして展開したものに含まれる 7.Alarm.pdf
(7) http://www.kc.t.u-tokyo.ac.jp/nl-resource/knp.html
(8) http://www.geocities.co.jp/SiliconValley-SanJose/5780/grammer.html
(9) 加賀野井 秀一；日本語は進化する，NHKブックス，2002.

ふじくら・としゆき (株)豆蔵

「VxWorks」を使ったRTOS技術の基礎と応用

第10回

組み込みシステムのデバッグ(中編)——VxWorksシミュレータ

高山 剛

シミュレータの有効活用

組み込みシステムを開発する場合，開発者の数だけターゲットのハードウェアを必要とします．場合によっては，一つはテスト用に，もう一つは開発用にと，一人で2〜3台も必要になってくることもあります．当然，たくさんのボードを購入したり試作するには，時間も経費もかかります．そこで，実ハードウェアがなくてもシミュレーションで代用できないかが検討されます．

一口にシミュレーションといっても，ハード/ソフトの協調開発・検証を行うHDLレベルのシミュレーションや，インストラクション・シミュレーションなどがあります．前者は，ミクロな視点でのハードウェアの動作確認や，簡単な実際のソフトウェアとの協調動作の検証デバッグに便利なようです．

インストラクション・シミュレータ(CPUの命令を一つずつ解析して実行するインタプリタ)は，原理的にはキャッシュのシミュレーションが可能で，キャッシュ・ヒット率を正確に求めることができるという利点があります．いずれもスーパコンピュータでも使わない限り，アプリケーションとOSを現実的なパフォーマンスで動作させることはできません．PCやワークステーションでは，OSとその上で動作する複雑なアプリケーションをデバッグするには動作速度が不十分でなかなか使いものになりません．

インストラクション・シミュレータは，コンパイラを移植する際のテストや，OSの移植の初期のフェーズでは有用でしょう．

最後のアプローチは，OSのAPIの互換性の高さに注目して，アプリケーションをPCやワークステーションのOS上でターゲットとなるOSをエミュレートし，ネイティブ・コードで高速に動作させる方法です．これはVxWorksシミュレータ(かつてはVxSimと呼ばれていた)で採用されている方法です．

図1(a)に実ターゲットを，図1(b)にVxWorksシミュレータを示します．Windows，Solaris上で一つのプロセス中にVxWorksのカーネルが存在していることに注目してください．UNIXのスレッドのように1プロセス内でスケジューラと複数のスレッドが存在するように，VxWorksシミュレータではVxWorksカーネルと複数のタスクが存在するイメージになります注1．

VxWorksのAPIはCPUアーキテクチャに依存せず，CPU間で完全に互換性があるため注2，シミュレータを用いれば，プロトタイプのアプリケーションをPentiumのコンパイラでコンパイルし，シミュレータ上で動作確認して，その後に実際のターゲットのCPUアーキテクチャのコンパイラで再コンパイルすることでソース・コードを修正せずに動作させることができます．

組み込みソフトウェアの開発現場では，ハードウェアとソフトウェアが同時に開発されることがあり，このような場面ではシミュレータを有効活用することができるでしょう．

VxWorksシミュレータの目的は，ハードウェアがなくてもソフトウェアのプロトタイプを作ることができる点にあります．また，プログラム・モジュールの単体テスト，テストの自動化

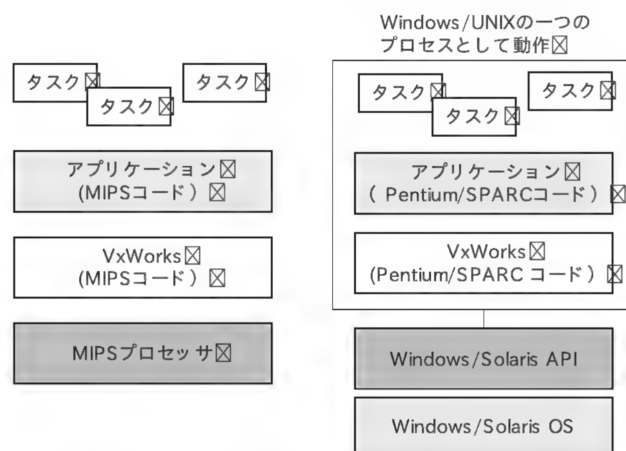


図1　実機のハードウェアとシミュレータのプロセスの比較

注1：UNIXやWindowsは仮想アドレスで動作するが，プロセス内はVxWorksと同じリニアなアドレス空間で成り立っている．
注2：VxWorksのソース・コードは，ただ一つリビジョンのソース・ツリーで構成されて，すべてのアーキテクチャをサポートしており，かつ，使用されているコンパイラのバージョンも同じであるため，高い互換性が実現されている．

も容易にできます．

ただし，シミュレータは，Ethernet，パラレルI/O，バスといったハードウェアをデバッグすることはできません．ハードウェアのビヘイビアを記述してハードウェアのエミュレーションを行うことはできますが，すべてのハードウェアのビヘイビア・モデルを網羅することは困難で，VxWorksシミュレータでは実現されていません．ドライバをそのまま動作させることができないため，ドライバの開発には使用できません．

あくまで，VxWorks上でユーザに公開された抽象度の高いAPIを使ったアプリケーションの開発とデバッグが可能です．ただし，後にも述べますが，ネットワーク，ファイル・システム，グラフィックス，タイマ，割り込み，マルチプロセッサのエミュレーションができるので，開発の多くの場面で使用できるでしょう．

なぜ，WindowsやSolaris上でVxWorksが動くのか？

図2は，実ターゲットのVxWorksとシミュレータのVxWorksの内部構造の比較です．カーネルやネットワーク・スタック，I/Oシステム，ANSI関数はアーキテクチャ非依存コードとしてコーディングされているので，実CPU版，シミュレータ版のVxWorksともまったく同じソース・コードで実現されています．

シミュレータの場合，CPUアーキテクチャ依存部分の一部をホストOSの機能で実現しています．たとえば，タスク・スイッチ時のコンテキスト切り替え時には，ホストOSのライブラリの`setjmp()/longjmp()`でCPUのレジスタ(コンテキスト)を切り替えています．このようにカーネルやスケジューラは実CPUを使ったVxWorksとまったく同一のコードを使用しているので，カーネルのふるまいは実CPU版のVxWorksと同じです．

ハードウェアのシミュレーションができないという問題は，

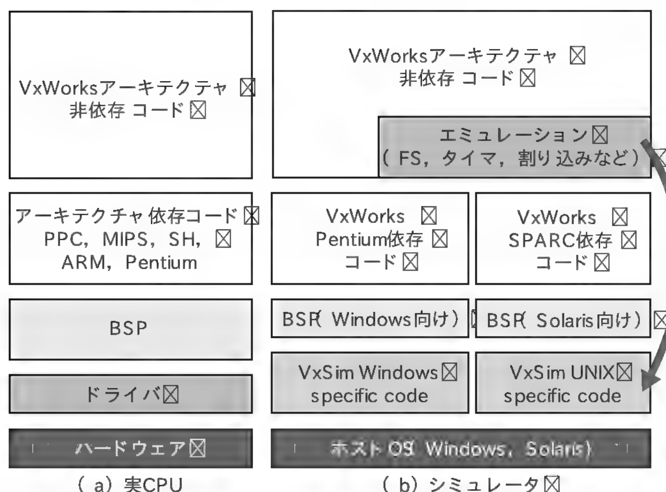


図2 シミュレータ版と実CPU版VxWorksの比較

ハードウェア依存性を隠蔽するAPIレベルでのエミュレーション，たとえばファイル・システム，タイマ，ネットワークなどのAPIをホストOSでエミュレーションによって実現し，ハードウェアはブラック・ボックスとしてホストOSの機能やリソースで代替させて実ターゲットと同じ環境を作り出すことで解決しています．

● ファイル・システムのエミュレーション

ファイル・システムはpassFS(Pass through FS)というシミュレータ専用の特殊なコンポーネントを使ってVxWorksのファイル・システムとの互換性を維持し，実際のファイルへのアクセスには，ブロック・デバイスを経由せず，ホストOSの`libc`に含まれる`open()`，`close()`，`read()`，`write()`へバイパスすることでファイル・システムをエミュレートしています．`libc`を使っているのでホストのファイル・システムがNFS(Network File System)マウントされていても問題ありません．

● タイマのエミュレーション

VxWorksのアプリケーションは，タイマに依存する機能(タスク・ディレイ，ウオッチドッグ・タイマなど)を使う場合が多く，タイマのエミュレーションが必須となります．

UNIXの場合，SIGALRMやSIGPROFといったシグナルを利用してホストOSから定期的にシグナルを受け取ることでエミュレーションを実現しています．もちろん，BSP(Board Support Package)によってデフォルトで設定されているので意識せず利用できます．

SIGALRM -> System Clock
SIGPROF -> Aux Clock(補助クロック)

● ネットワークのエミュレーション

ネットワークには，UNIXの場合はPPP，WindowsではULIP(User Level Internet Protocol)と呼ばれるドライバによってIPレベルでのエミュレーションが可能になっています(詳細は，後述のネットワーク・アプリケーション参照)．

● 割り込みのエミュレーション

VxWorksシミュレータは，ハードウェア依存のアプリケーション開発には向きませんが，どうしても割り込みのエミュレーションが必要な場合のために，UNIXのシグナルを送信して，疑似的に割り込みを発生させることができます．シミュレータ側で割り込みを受け取る場合，CPU外部割り込みをハンドリングするのと同様に，`intConnect()`を使って次のようなコーディングを行います．

```
intConnet ( 17 , logMsg ,“ Interrupt!¥n” );
```

`intConnect()`の最初の引き数は割り込みベクタですが，シミュレータの場合は，ユーザ定義のシグナル番号(Windowsの場合はWindows message)を設定します．

UNIX場合，

```
SIGUSR2  16
SIGUSR2  17
```

が，Windowsの場合，

```
0xc011
```

が利用できます．

シグナルを発生させる場合，UNIXでは次のように呼び出します．

```
UNIX->kill signal pid
```

シミュレータでの割り込みは，インタラクティブに任意のタイミングでアプリケーションの流れを変えたい場合など，ハードウェアのエミュレーションが可能かもしれません．

シミュレータのVxWorksは，このように各コンポーネントで相当するホストOSのシステム・コールを呼び出しています．したがって，ホストOSの応答性に依存しているため，リアルタイム応答性はもちません．あくまで非リアルタイム・アプリケーションの開発環境に使うことになります．



図3　Wind MLの構成

VxWorksエミュレータのパフォーマンスに関して

前述したとおり，VxWorksシミュレータはホストOS上でネイティブ・コードで動作します．したがって，システム・コールを含まない場合は，CPUの性能がそのまま発揮できることを意味します．しかし，VxWorksのAPIを呼び出した場合，ホストOS上でエミュレーションを行うので，どれだけ実行時間がかかるかはホストOSに依存することになります．

たとえば，割り込みを禁止する`intLock()`は，実ターゲットで動作するVxWorksではCPUのステータス・レジスタにビットをセットするだけなので非常に高速ですが，シミュレータでは，ホストOSにシステム・コールを行い，シグナルのマスクを行うので，けっこうなオーバヘッドをともなうことになります．

また，VxWorksシミュレータはホストOSの1プロセスとして動作するので，ホストOSのプロセスのスケジューリングにも影響されます．さらに，ホストOSによりスワップやページングが発生することがあるので，リアルタイム性やプリエンプティブ性[注3]はまったく期待できないことに留意する必要があります．

Wind MLのサポート

VxWorksには，Wind ML(Media Library)というグラフィック・ライブラリが提供されています[注4]．Wind ML(図3)は2Dグラフィックス，オーバレイ，アルファ・ブレンディング，アンチエイリアシング，フォント・エンジン，オーディオ，ポインティング・デバイス，マルチウィンドウなどの機能をもち，スケーラブルで非常にコンパクトなコード・サイズ(100～200Kバイト)という特徴をもったVxWorksのコンポーネント(ミドルウェア)です．グラフィック・ドライバにVxWorksシミュレータ用のドライバをサポートしています．

グラフィック・ドライバにVxWorksシミュレータ用のグラフィック・ドライバ(Solaris用，Windows用がある)を選択することで，VxWorksシミュレータでグラフィック・アプリケーションを開発することができます．

図4は，実際にシミュレータ上にNetFront((株)ACCESS)を移植[注5]した際のブロック図です(エンジニアによって検証用に移植が試みられただけなので，製品としては販売されていない)．

図5，写真1は，NetFrontを実際のターゲット・ハードウェア上[注6]に移植したものです．

図4，図5は，それぞれで動作させた場合のソフトウェア・ブロック図です．図に示すように，アプリケーションのNetFrontがWind ML上でハードウェアに依存せずに動作し，シミュレータと実ターゲットのハードウェアの違いはWind MLの下位レイヤやULIPドライバで吸収されています．実ターゲットでは，マウスやキーボードにUSBを，無線LANやグラフィクスにPCIカードを使用していますが，VxWorksシミュレータ用のグラフィック・ドライバがWindows上のウィンドウと

注3：どのような状況でも，もっとも高いプライオリティをもつタスクがCPUを占有していること．VxWorksシミュレータ自体はプリエンプティブ性をもっているが，VxWorksシミュレータとほかのWindowsアプリケーションの間ではプリエンプティブ性はない．

注4：WIND MLはVxWorksに含まれているコンポーネントではないが，VxWorksをコアにしたディジタル・コンシューマ向けプラットホーム製品などに含まれている．

注5：アイティアクセス(株) http://www.itaccess.co.jp/　info@itaccess.co.jpによる．

注6：東芝セミコンダクター社製CPU TX4938とコンパニオン・チップGOKU-Sをベースとし，NANDフラッシュ・メモリ，PCI，USB，Wired & Wireless LANを組み合わせたディジタル・テレビやセットトップ・ボックスに代表されるディジタル家電向けのマルチメディア・プラットホーム(http://www.windriver.com/japan/alliances/directory/list/toshiba/index.html)．

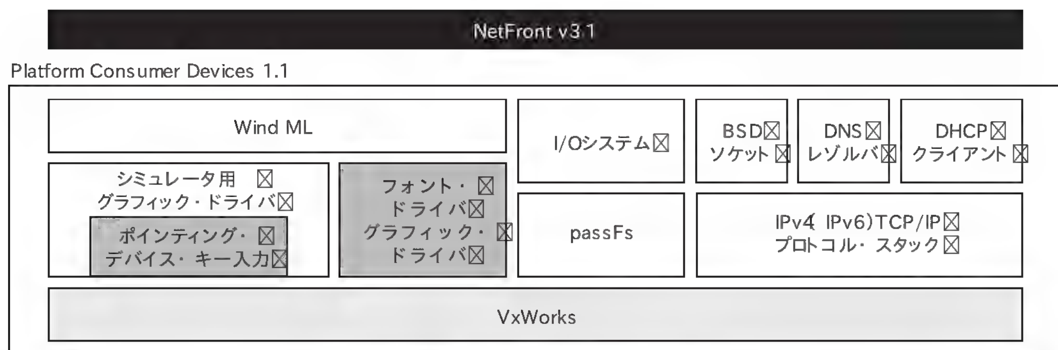


図4
シミュレータ上での NetFront

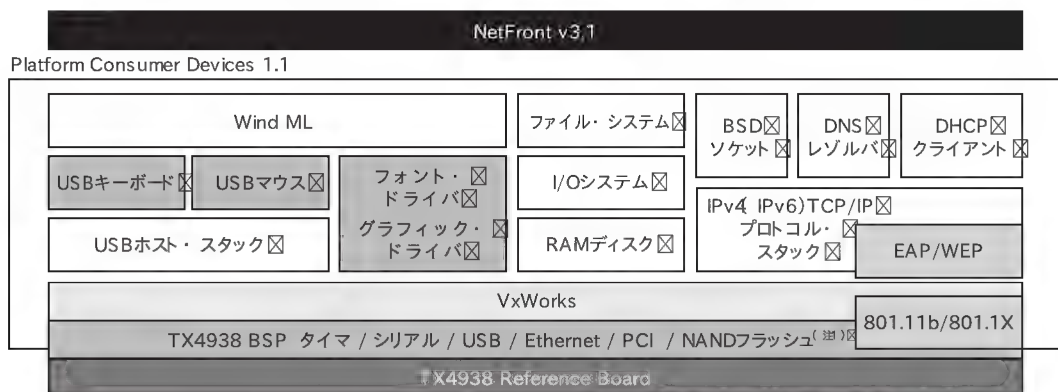


図5
実ターゲットでの NetFront

注）サポート外のサンプル・ドライバによる（DosFs対応）

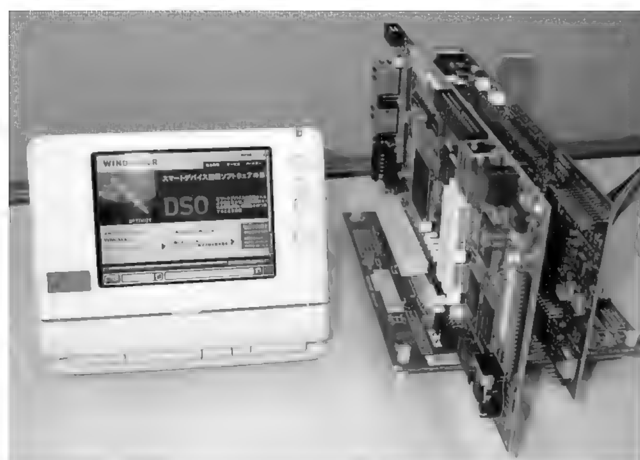


写真1　実ターゲット上での NetFront



図6　Tornado からシミュレータを起動—シミュレータ上で NetFront を動かす

カーソル/キーボードを，ULIPドライバがEthernetをエミュレートしています．

シミュレータから ULIPドライバを介して送信（受信）されるIPパケットをホスト OSがルーティングすることで，インターネットへもアクセスできます．図6では，シミュレータ上のNetFrontからホスト OSをゲートウェイとしてインターネットへアクセスしています．VxWorksシミュレータのエミュレーションをフルに生かした実例といえます．

ネットワーク・アプリケーションでの対応

VxWorksシミュレータが自身のIPアドレスをもって，イン

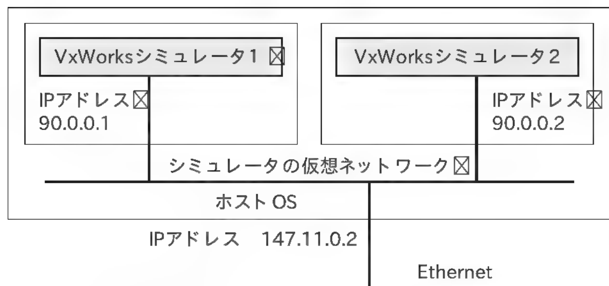


図7　複数のインスタンス

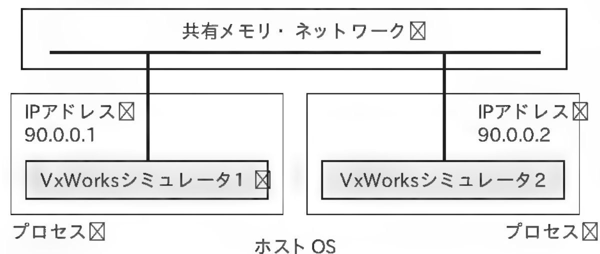


図8　共有メモリ・ネットワークのエミュレーション

ターネットへのアクセス，またほかのVxWorksシミュレータや実ターゲット(VxWorks，あるいはそのほかのOS)とTCP/IPやUDPで通信が可能です．このことは，FTPやNFS，RPCといったSocketやTCP/IP，UDPベースのアプリケーションがそのまま動作することを意味します．

具体的なエミュレーションの方法は，SolarisマシンではSolarisのPPPコンポーネントを使い，WindowsマシンではWind Riverが提供するULIPドライバをNDIS(Windowsのネットワーク・デバイス・ドライバのインターフェース)としてWindowsにインストールすることで，シミュレータに仮想のネットワーク・インターフェースをもたせることで実現しています注7．

VxWorksシミュレータは複数のインスタンスを，図7のようにホストOSの一つのプロセスとして起動させることができます．このままではお互いは通信はできませんが，PPP(Point to Point Protocol)やULIPを使うことでTCP/UDP/IPレベルでのマルチプロセッサ構成のアプリケーションの開発，デバッグができます．

マルチプロセッサ・システムのエミュレーションとデバッグ環境

VxWorksは，バックプレーン(VMEバスなど)を介して複数のCPUボードを使用した組み込みシステムでも使用されます．

注7：Soalris(UNIX)ではPPPを使うので，シリアル・インターフェースの先にSolarisホストが存在しているイメージ．

注8：通信相手を指定する必要がないこと，通信に失敗することが100%ないため．

Column
VxWorks6.0のメモリ・プロテクション

次世代のVxWorks6.0では，まったく新しい考え方のメモリ・プロテクションが導入されていますが，VxWorks6.0のシミュレータはメモリ・プロテクションに対応しています．一般的に，UNIXの一つのプロセスの中で，UNIXカーネル自体をエミュレートすることはUNIXが論理アドレスを重複してもつため無理と考えられますが，VxWorks6.0では，UNIXとは異なり，論理アドレス空間は重複せず，それぞれの論理アドレス空間は，システム中でユニークなアドレス空間を持つ構造になっています．

このため，UNIX，Windows上でMMU機能をフルに使っているVxWorks6.0カーネルのエミュレーションが可能になっています．

この場合，VMEバスのマルチマスタ機能を最大限に利用するために，VxMPという共有メモリ方式の密結合のCPU間通信や，同期が可能なミドルウェアがありますが，VxWorksシミュレータでも共有メモリ・ネットワークをエミュレーションにより実現しているので密結合型のマルチプロセッサのアプリケーションもシミュレータで開発が可能です(図8)．

VxWorksシミュレータはホストOSで動作し，リアルタイム性をもたないことを除いて，実CPU上で動作するVxWorksと同じなので，実CPU版のVxWorksとまったく同じ開発環境，デバッグ環境，ツールを使用することができます．

さらに，実ターゲットではIPアドレスの設定などを最低限行う必要がありますが，VxWorksシミュレータの場合は，ホストOS上に開発環境もシミュレータも同一のマシンで動作するので，通信方法にUNIXパイプを使う方法を追加しています．UNIXパイプにより設定方法がさらに簡略化されています注8．

おわりに

VxWorksシミュレータは以前は，オプション製品で別途購入が必要でした．現在は，VxWorksやプラットホーム製品に付属しているので，正規にライセンスを受けていれば実CPU版もVxWorksシミュレータも両方とも使えます．実際のところ，組み込みソフトウェア開発では実機がないと，厳密なデバッグはできませんが，もう1～2台実機があればテストとちょっとした動作確認が同時並列で効率良くできるのにと思う瞬間はないでしょうか．こんなときにシミュレータを使ってみると便利でしょう．

参考文献
(1) Tornado 2.1.1 User's Guide

たかやま・たけし　ウインドリバー(株)



第20回

GCC2.95から追加変更のあったオプションの補足と検証(その8)

岸 哲夫

今回も引き続きGCC2.95から追加変更のあったオプションの補足と検証を行う．今回は，「最適化オプション」について扱う．最適化オプションには，ここで紹介するほかにも“SSA”(Static Single Assignment)の試験サポートがある．これに関しては最適化オプションの最後に説明する．（筆者）

● -fnew-ra

このオプションは試験的に提供されたものですが，どうやら失敗作だったようです．処理を遅くするだけのコードを生成するようで，評価は良くありません．これを再度評価する意味もなさそうなので，検証は行いません．

● -fno-guess-branch-probability

分岐予測による最適化を行わず，同一のソースをコンパイルしたときに，つねに同じアセンブラ・コードを生成するように指定するオプションです．分岐の予測を行う最適化が行われた際，場合によっては同じソースから別のコードが生成されることがあります．それが組み込みシステムの設計に影響を及ぼすこともあるので，それを防止します．

● -fno-inline

キーワード inline を無視します．このオプションは，関数のインライン展開をいっさい行わせないために使われます．メリットは，メモリが少ないという理由で実行コードが不安定な場合，実行が安定するという点です．

オプションの指定の方法は次のとおりです．

●指定なし

```
gcc test226.c -S  -finline
```

●指定あり

```
gcc test226.c -S  -fno-inline
```

ソースと生成されたコードを，リスト1～リスト5に示します．

単純に a をインクリメントしている関数 test1 は -finline

リスト1 -fno-inlineオプションを使う例(test226.c)

```
//インライン・ファンクションの例
#include <stdio.h>
static int    a;
static inline int test1();
static inline int test1()
{
    return    a++;
}
int main()
{
    int       res;
    a    =    1;
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    return    0;
}
```

リスト2 -fno-inlineオプションを付けて生成されたアセンブラ・ソース(test226a.s)

```
    .file     "test226.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $1, a
    movl      a, %eax
    incl      a
    movl      %eax, -8(%ebp)
    movl      -8(%ebp), %eax
    movl      %eax, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    movl      a, %eax
    incl      a
    movl      %eax, -8(%ebp)
    movl      -8(%ebp), %eax
    movl      %eax, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
    .local    a
    .comm     a,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 3　-finline オプションを付けて生成されたアセンブラ・ソース(test226.s)

```
    .file     "test226.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $1, a
    call      test1
    movl      %eax, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    call      test1
    movl      %eax, -4(%ebp)
    subl      $8, %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
    .local    a
    .comm     a,4,4
    .type     test1, @function
test1:
    pushl     %ebp
    movl      %esp, %ebp
    movl      a, %eax
    incl      a
    leave
    ret
    .size     test1, .-test1
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 4　-fno-inline オプションを付けて生成されたシンボル・リスト(test226a.nm)

```
080494d0 D _DYNAMIC
0804959c D _GLOBAL_OFFSET_TABLE_
080484b0 R _IO_stdin_used
         w _Jv_RegisterClasses
080494c0 d __CTOR_END__
080494bc d __CTOR_LIST__
080494c8 d __DTOR_END__
080494c4 d __DTOR_LIST__
080484b8 r __FRAME_END__
080494cc d __JCR_END__
080494cc d __JCR_LIST__
080495bc A __bss_start
080495b0 D __data_start
0804846c t __do_global_ctors_aux
08048308 t __do_global_dtors_aux
080495b4 D __dso_handle
080494bc A __fini_array_end
080494bc A __fini_array_start
         w __gmon_start__
080494bc A __init_array_end
080494bc A __init_array_start
08048428 T __libc_csu_fini
080483e0 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080494bc A __preinit_array_end
080494bc A __preinit_array_start
080495bc A _edata
080495c4 A _end
08048490 T _fini
080484ac R _fp_hw
08048278 T _init
080482c0 T _start
080495c0 b a
080482e4 t call_gmon_start
080495bc b completed.1
080495b0 W data_start
08048344 t frame_dummy
08048370 T main
080495b8 d p.0
         U printf@@GLIBC_2.0
```

リスト 5　-finline オプションを付けて生成されたシンボル・リスト(test226.nm)

```
080494c8 D _DYNAMIC
08049594 D _GLOBAL_OFFSET_TABLE_
080484a8 R _IO_stdin_used
         w _Jv_RegisterClasses
080494b8 d __CTOR_END__
080494b4 d __CTOR_LIST__
080494c0 d __DTOR_END__
080494bc d __DTOR_LIST__
080484b0 r __FRAME_END__
080494c4 d __JCR_END__
080494c4 d __JCR_LIST__
080495b4 A __bss_start
080495a8 D __data_start
08048464 t __do_global_ctors_aux
08048308 t __do_global_dtors_aux
080495ac D __dso_handle
080494b4 A __fini_array_end
080494b4 A __fini_array_start
         w __gmon_start__
080494b4 A __init_array_end
080494b4 A __init_array_start
08048420 T __libc_csu_fini
080483d8 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080494b4 A __preinit_array_end
080494b4 A __preinit_array_start
080495b4 A _edata
080495bc A _end
08048488 T _fini
080484a4 R _fp_hw
08048278 T _init
080482c0 T _start
080495b8 b a
080482e4 t call_gmon_start
080495b4 b completed.1
080495a8 W data_start
08048344 t frame_dummy
08048370 T main
080495b0 d p.0
         U printf@@GLIBC_2.0
080483c7 t test1
```

リスト 6　関数名が重複している例1(test227.c)

```
//インライン・ファンクションの例
int test1();
int test2();
int test1()
{
    int  a    =    100;
    return    a++;
}
int test2()
{
    int  b    =    test1();
    return    b;
}
```

リスト 8　関数名が重複している例3(test229.c)

```
//インライン・ファンクションの例
static   int  test1();
int test2();
static   int  test1()
{
    int  a    =    100;
    return    a++;
}
int test2()
{
    int  b    =    test1();
    return    b;
}
```

ではソース中に展開されていますが，-fno-inlineの指定では関数として存在しています．

たとえば，古いソースと新しいソースをリンクしたいときに古いソースの中に関数「test1」が存在して，新しいソースの中にも関数「test1」が存在している場合には，新しいソース中の関数にinlineキーワードを付加し，オプション-finlineを付けることによってシンボル・リストからも名前を消すことができ，正常にリンクすることが可能です．

リスト 6とリスト 7の例ではエラーになります．コンパイル時のようすを下記に示します．

```
$ gcc test228.c test227.c -o test227.o
                                  -finline
/tmp/cc8cJJqr.o(.text+0x0)
                    : In function `test1':
: multiple definition of `test1'
/tmp/ccSOBchX.o(.text+0x0)
                     : first defined here
/usr/bin/ld: Warning: size of symbol
  `test1' changed from 16 in /tmp/ccSOBchX.o
                    to 23 in /tmp/cc8cJJqr.o
collect2: ld  1
$
```

しかし，リスト 7とリスト 8の例では問題ありません．コンパイル時および実行時のようすを下記に示します．

```
$ gcc test228.c test229.c -o test227
$ ./test227
```

リスト 7　関数名が重複している例2(test228.c)

```
//インライン・ファンクションの例
#include <stdio.h>
static int    a;
int test1();
int test1()
{
    return    a++;
}
int main()
{
    int       res;
    a    =    1;
    res  =    test1();
    printf("%d¥n",res);
    res  =    test1();
    printf("%d¥n",res);
    res  =    test2();
    printf("%d¥n",res);
    return    0;
}
```

```
1
2
100
$
```

リスト 9に示すオブジェクト・ダンプを見ると，二つの関数test1は別のアドレスとして認識されています．test229.c中のtest1はローカル関数として扱われています．

```
$ gcc test228.c test229.c -o test227 -O3
$ objdump -D test227 > test227a.txt
```

最適化オプション-O3を付加すると，ソース中の関数を効率が上がるならばインライン展開するように最適化します(リスト 10)．

その結果，test228.c中の関数test1をインライン展開しています．

インライン展開は，ソース単位のコードに対して行われます．つまり，extern関数はインライン展開されません．関数に__inlineを指定した場合，ほかのソースからは呼び出せなくなります．

● -fno-math-errno

このオプションを指定すると数値演算後のエラー・チェックを行いません．よってループ中にそのコードがあった場合には，指定すると多少速くなるでしょう．

● -fno-peephole

このオプションを指定すると「のぞき穴」最適化を行いません．-fno-peephole2も同様です．マシン固有の最適化や，比較的細かい最適化を行わないようにします．この最適化によって，冗長な処理または変数が排除された場合に，このコードとリンクした別の単位のコードに影響をおよぼすおそれがあります．そんなときは冗長なコードを排除しないように，このオプションを付ければ問題は起きません．

リスト 11を例にして最適化のようすを示します．

```
$gcc test230.c -O -S
```

リスト 9 生成されたオブジェクト・ダンプ・リスト(test227.txt)

```
test227:     ファイル形式 elf32-i386

セクション .interp の逆アセンブル:

08048114 <.interp>:
    ----------略----------
セクション .note.ABI-tag の逆アセンブル:

08048128 <.note.ABI-tag>:
    ----------略----------
    ...
セクション .hash の逆アセンブル:

08048148 <.hash>:
    ----------略----------
    ...
セクション .dynsym の逆アセンブル:

08048174 <.dynsym>:
    ...
    ----------略----------
    ...
セクション .dynstr の逆アセンブル:

080481d4 <.dynstr>:
    ----------略----------
セクション .gnu.version の逆アセンブル:

08048234 <.gnu.version>:
 8048234:     00 00                    add    %al,(%eax)
 8048236:     02 00                    add    (%eax),%al
 8048238:     02 00                    add    (%eax),%al
 804823a:     01 00                    add    %eax,(%eax)
 804823c:     00 00                    add    %al,(%eax)
    ...
セクション .gnu.version_r の逆アセンブル:

08048240 <.gnu.version_r>:
    ----------略----------
    ...
セクション .rel.dyn の逆アセンブル:

08048260 <.rel.dyn>:
 8048260:     d8 95 04 08 06 05        fcoms  0x5060804(%ebp)
    ...
セクション .rel.plt の逆アセンブル:

08048268 <.rel.plt>:
    ----------略----------
    ...
セクション .init の逆アセンブル:

08048278 <_init>:
    ----------略----------
セクション .plt の逆アセンブル:

08048290 <.plt>:
    ----------略----------
セクション .text の逆アセンブル:

080482c0 <_start>:
    ----------略----------

080482e4 <call_gmon_start>:
    ----------略----------

08048308 <__do_global_dtors_aux>:
    ----------略----------
<__do_global_dtors_aux+0x38>
    ----------略----------
<__do_global_dtors_aux+0x31>
    ----------略----------
<__do_global_dtors_aux+0x1c>
    ----------略----------

08048344 <frame_dummy>:
    ----------略----------

08048370 <test1>:
 8048370:     55                       push   %ebp
 8048371:     89 e5                    mov    %esp,%ebp
 8048373:     a1 00 96 04 08           mov    0x8049600,%eax
 8048378:     ff 05 00 96 04 08        incl   0x8049600
 804837e:     c9                       leave
 804837f:     c3                       ret

08048380 <main>:
 8048380:     55                       push   %ebp
 8048381:     89 e5                    mov    %esp,%ebp
 8048383:     83 ec 08                 sub    $0x8,%esp
 8048386:     83 e4 f0                 and    $0xfffffff0,%esp
 8048389:     b8 00 00 00 00           mov    $0x0,%eax
 804838e:     29 c4                    sub    %eax,%esp
 8048390:     c7 05 00 96 04 08 01     movl   $0x1,0x8049600
 8048397:     00 00 00
 804839a:     e8 d1 ff ff ff           call   8048370 <test1>
 804839f:     89 45 fc                 mov    %eax,0xfffffffc(%ebp)
 80483a2:     83 ec 08                 sub    $0x8,%esp
 80483a5:     ff 75 fc                 pushl  0xfffffffc(%ebp)
 80483a8:     68 f4 84 04 08           push   $0x80484f4
 80483ad:     e8 fe fe ff ff           call   80482b0 <_init+0x38>
 80483b2:     83 c4 10                 add    $0x10,%esp
 80483b5:     e8 b6 ff ff ff           call   8048370 <test1>
 80483ba:     89 45 fc                 mov    %eax,0xfffffffc(%ebp)
 80483bd:     83 ec 08                 sub    $0x8,%esp
 80483c0:     ff 75 fc                 pushl  0xfffffffc(%ebp)
 80483c3:     68 f4 84 04 08           push   $0x80484f4
 80483c8:     e8 e3 fe ff ff           call   80482b0 <_init+0x38>
 80483cd:     83 c4 10                 add    $0x10,%esp
 80483d0:     e8 36 00 00 00           call   804840b <test2>
 80483d5:     89 45 fc                 mov    %eax,0xfffffffc(%ebp)
 80483d8:     83 ec 08                 sub    $0x8,%esp
 80483db:     ff 75 fc                 pushl  0xfffffffc(%ebp)
 80483de:     68 f4 84 04 08           push   $0x80484f4
 80483e3:     e8 c8 fe ff ff           call   80482b0 <_init+0x38>
 80483e8:     83 c4 10                 add    $0x10,%esp
 80483eb:     b8 00 00 00 00           mov    $0x0,%eax
 80483f0:     c9                       leave
 80483f1:     c3                       ret
 80483f2:     90                       nop
 80483f3:     90                       nop

080483f4 <test1>:
 80483f4:     55                       push   %ebp
 80483f5:     89 e5                    mov    %esp,%ebp
 80483f7:     83 ec 04                 sub    $0x4,%esp
 80483fa:     c7 45 fc 64 00 00 00     movl
                                             $0x64,0xfffffffc(%ebp)
 8048401:     8b 45 fc                 mov    0xfffffffc(%ebp),%eax
 8048404:     8d 55 fc                 lea    0xfffffffc(%ebp),%edx
 8048407:     ff 02                    incl   (%edx)
 8048409:     c9                       leave
 804840a:     c3                       ret

0804840b <test2>:
 804840b:     55                       push   %ebp
 804840c:     89 e5                    mov    %esp,%ebp
 804840e:     83 ec 08                 sub    $0x8,%esp
 8048411:     e8 de ff ff ff           call   80483f4 <test1>
 8048416:     89 45 fc                 mov    %eax,0xfffffffc(%ebp)
 8048419:     8b 45 fc                 mov    0xfffffffc(%ebp),%eax
 804841c:     c9                       leave
 804841d:     c3                       ret
 804841e:     90                       nop
 804841f:     90                       nop

08048420 <__libc_csu_init>:
    ----------略----------

08048468 <__libc_csu_fini>:
    ----------略----------

080484ac <__do_global_ctors_aux>:
    ----------略----------
<__do_global_ctors_aux+0x20>
    ----------略----------
<__do_global_ctors_aux+0x14>
    ----------略----------
セクション .fini の逆アセンブル:

080484d0 <_fini>:
    ----------略----------
セクション .rodata の逆アセンブル:
```

リスト 9　生成されたオブジェクト・ダンプ・リスト 1（test227.txt）（つづき）

```
080484ec <_fp_hw>:
 80484ec:     03 00                    add    (%eax),%eax
    ...

080484f0 <_IO_stdin_used>:
    ----------略----------
セクション .eh_frame の逆アセンブル:

080484f8 <__FRAME_END__>:
 80484f8:     00 00                    add    %al,(%eax)
    ...
セクション .ctors の逆アセンブル:

080494fc <__CTOR_LIST__>:
    ----------略----------

08049500 <__CTOR_END__>:
 8049500:     00 00                    add    %al,(%eax)
    ...
セクション .dtors の逆アセンブル:

08049504 <__DTOR_LIST__>:
    ----------略----------

08049508 <__DTOR_END__>:
 8049508:     00 00                    add    %al,(%eax)
    ...
セクション .jcr の逆アセンブル:

0804950c <__JCR_END__>:
 804950c:     00 00                    add    %al,(%eax)
    ...
セクション .dynamic の逆アセンブル:

08049510 <_DYNAMIC>:
    ----------略----------
    ...
セクション .got の逆アセンブル:

080495d8 <.got>:
 80495d8:     00 00                    add    %al,(%eax)
    ...
セクション .got.plt の逆アセンブル:

080495dc <_GLOBAL_OFFSET_TABLE_>:
    ----------略----------
セクション .data の逆アセンブル:

080495f0 <__data_start>:
 80495f0:     00 00                    add    %al,(%eax)
    ...

080495f4 <__dso_handle>:
 80495f4:     00 00                    add    %al,(%eax)
    ...

080495f8 <p.0>:
 80495f8:     08                       .byte 0x8
 80495f9:     95                       xchg   %eax,%ebp
 80495fa:     04 08                    add    $0x8,%al
セクション .bss の逆アセンブル:

080495fc <completed.1>:
 80495fc:     00 00                    add    %al,(%eax)
    ...

08049600 <a>:
 8049600:     00 00                    add    %al,(%eax)
    ...
セクション .comment の逆アセンブル:

00000000 <.comment>:
    ----------略----------
    ...
```

リスト 10　生成されたオブジェクト・ダンプ・リスト 2（test227a.txt）

```
test227:     ファイル形式 elf32-i386

セクション .interp の逆アセンブル:

08048114 <.interp>:
    ----------略----------
セクション .note.ABI-tag の逆アセンブル:

08048128 <.note.ABI-tag>:
    ----------略----------
    ...
セクション .hash の逆アセンブル:

08048148 <.hash>:
    ----------略----------
    ...
セクション .dynsym の逆アセンブル:

08048174 <.dynsym>:
    ...
    ----------略----------
    ...
セクション .dynstr の逆アセンブル:

080481d4 <.dynstr>:
    ----------略----------
セクション .gnu.version の逆アセンブル:

08048234 <.gnu.version>:
    ----------略----------
    ...
セクション .gnu.version_r の逆アセンブル:

08048240 <.gnu.version_r>:
    ----------略----------
    ...
セクション .rel.dyn の逆アセンブル:

08048260 <.rel.dyn>:
    ----------略----------
    ...
セクション .rel.plt の逆アセンブル:

08048268 <.rel.plt>:
    ----------略----------
    ...
セクション .init の逆アセンブル:

08048278 <_init>:
    ----------略----------
セクション .plt の逆アセンブル:

08048290 <.plt>:
    ----------略----------
セクション .text の逆アセンブル:

080482c0 <_start>:
    ----------略----------

080482e4 <call_gmon_start>:
    ----------略----------

08048308 <__do_global_dtors_aux>:
    ----------略----------
<__do_global_dtors_aux+0x38>
    ----------略----------
```

リスト 10　生成されたオブジェクト・ダンプ・リスト 2（test227a.txt）（つづき）

```
<__do_global_dtors_aux+0x31>
     ----------略----------
<__do_global_dtors_aux+0x1c>
     ----------略----------

08048344 <frame_dummy>:
     ----------略----------

08048370 <main>:
 8048370:     55                      push   %ebp
 8048371:     89 e5                   mov    %esp,%ebp
 8048373:     83 ec 08                sub    $0x8,%esp
 8048376:     83 e4 f0                and    $0xfffffff0,%esp
 8048379:     83 ec 08                sub    $0x8,%esp
 804837c:     6a 01                   push   $0x1
 804837e:     68 b8 84 04 08          push   $0x80484b8
 8048383:     c7 05 c4 95 04 08 02    movl   $0x2,0x80495c4
 804838a:     00 00 00
 804838d:     e8 1e ff ff ff          call   80482b0 <_init+0x38>
 8048392:     58                      pop    %eax
 8048393:     8b 0d c4 95 04 08       mov    0x80495c4,%ecx
 8048399:     5a                      pop    %edx
 804839a:     51                      push   %ecx
 804839b:     8d 41 01                lea    0x1(%ecx),%eax
 804839e:     68 b8 84 04 08          push   $0x80484b8
 80483a3:     a3 c4 95 04 08          mov    %eax,0x80495c4
 80483a8:     e8 03 ff ff ff          call   80482b0 <_init+0x38>
 80483ad:     e8 26 00 00 00          call   80483d8 <test2>
 80483b2:     5a                      pop    %edx
 80483b3:     59                      pop    %ecx
 80483b4:     50                      push   %eax
 80483b5:     68 b8 84 04 08          push   $0x80484b8
 80483ba:     e8 f1 fe ff ff          call   80482b0 <_init+0x38>
 80483bf:     31 c0                   xor    %eax,%eax
 80483c1:     c9                      leave
 80483c2:     c3                      ret
 80483c3:     90                      nop

080483c4 <test1>:
 80483c4:     a1 c4 95 04 08          mov    0x80495c4,%eax
 80483c9:     55                      push   %ebp
 80483ca:     89 e5                   mov    %esp,%ebp
 80483cc:     8d 48 01                lea    0x1(%eax),%ecx
 80483cf:     89 0d c4 95 04 08       mov    %ecx,0x80495c4
 80483d5:     c9                      leave
 80483d6:     c3                      ret
 80483d7:     90                      nop

080483d8 <test2>:
 80483d8:     55                      push   %ebp
 80483d9:     89 e5                   mov    %esp,%ebp
 80483db:     b8 64 00 00 00          mov    $0x64,%eax
 80483e0:     c9                      leave
 80483e1:     c3                      ret
 80483e2:     90                      nop
 80483e3:     90                      nop

080483e4 <__libc_csu_init>:
     ----------略----------

0804842c <__libc_csu_fini>:
     ----------略----------

08048470 <__do_global_ctors_aux>:
     ----------略----------
<__do_global_ctors_aux+0x20>
     ----------略----------
<__do_global_ctors_aux+0x14>
     ----------略----------
セクション .fini の逆アセンブル:

08048494 <_fini>:
     ----------略----------
セクション .rodata の逆アセンブル:

080484b0 <_fp_hw>:
 80484b0:     03 00                   add    (%eax),%eax
     ...

080484b4 <_IO_stdin_used>:
     ----------略----------
セクション .eh_frame の逆アセンブル:

080484bc <__FRAME_END__>:
 80484bc:     00 00                   add    %al,(%eax)
     ...
セクション .ctors の逆アセンブル:

080494c0 <__CTOR_LIST__>:
     ----------略----------

080494c4 <__CTOR_END__>:
 80494c4:     00 00                   add    %al,(%eax)
     ...
セクション .dtors の逆アセンブル:

080494c8 <__DTOR_LIST__>:
     ----------略----------

080494cc <__DTOR_END__>:
 80494cc:     00 00                   add    %al,(%eax)
     ...
セクション .jcr の逆アセンブル:

080494d0 <__JCR_END__>:
 80494d0:     00 00                   add    %al,(%eax)
     ...
セクション .dynamic の逆アセンブル:

080494d4 <_DYNAMIC>:
     ----------略----------
     ...
セクション .got の逆アセンブル:

0804959c <.got>:
 804959c:     00 00                   add    %al,(%eax)
     ...
セクション .got.plt の逆アセンブル:

080495a0 <_GLOBAL_OFFSET_TABLE_>:
     ----------略----------
セクション .data の逆アセンブル:

080495b4 <__data_start>:
 80495b4:     00 00                   add    %al,(%eax)
     ...

080495b8 <__dso_handle>:
 80495b8:     00 00                   add    %al,(%eax)
     ...

080495bc <p.0>:
     ----------略----------
セクション .bss の逆アセンブル:

080495c0 <completed.1>:
 80495c0:     00 00                   add    %al,(%eax)
     ...

080495c4 <a>:
 80495c4:     00 00                   add    %al,(%eax)
     ...
セクション .comment の逆アセンブル:

00000000 <.comment>:
     ----------略----------
```

リスト 11「覗き穴」最適化の例(test230.c)

```
//ピープホール最適化を行う例
#include <stdio.h>
main()
{
    int  x    =    0;
    int  y    =    0;
    x    =    x    +    1;
    y    =    x^2;
    printf("%d¥n",x);
    printf("%d¥n",y);
    y    =    y    +    0;
    y    =    y    +    0;
    y    =    y    +    0;
    y    =    y    +    0;
    y    =    y    +    0;
    goto L1;
    return;
L1:
    return;
    printf("%d¥n",x);
}
```

リスト 12に，最適化-Oで生成されたアセンブラ・ソースを示します．

$gcc test230.c -fno-peephole -S

リスト 13に，「のぞき穴」最適化を禁止して生成されたアセンブラ・ソースを示します．

$gcc test230.c -O3 -S

リスト 14に，最適化-O3で生成されたアセンブラ・ソースを示します．

このように，リスト 13には冗長なコードが残っています．最適化を強くすればするほど冗長なコードは整理されます．

● -fno-sched-interblock

ターゲット・マシン上でこのフラグがサポートされている場

リスト 12　最適化-Oで生成されたアセンブラ・ソース(test230a.s)

```
    .file     "test230.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8,   %esp
    andl      $-16, %esp
    subl      $8,   %esp
    pushl     $1
    pushl     $.LC0
    call      printf
    addl      $8,  %esp
    pushl     $3
    pushl     $.LC0
    call      printf
    addl      $16, %esp
.L2:
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 13「のぞき穴」最適化を禁止して生成されたアセンブラ・ソース(test230b.s)

```
    .file     "test230.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp,      %ebp
    subl      $8,        %esp
    andl      $-16,      %esp
    movl      $0,        %eax
    subl      %eax,      %esp
    movl      $0,        -4(%ebp)
    movl      $0,        -8(%ebp)
    leal      -4(%ebp), %eax
    incl      (%eax)
    movl      -4(%ebp), %eax
    xorl      $2,    %eax
    movl      %eax, -8(%ebp)
    subl      $8,    %esp
    pushl     -4(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16,   %esp
    subl      $8,    %esp
    pushl     -8(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16,   %esp
.L2:
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 14　最適化-O3で生成されたアセンブラ・ソース(test230.s)

```
    .file     "test230.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%d¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main, @function
main:
.L2:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8,   %esp
    andl      $-16, %esp
    subl      $8,   %esp
    pushl     $1
    pushl     $.LC0
    call      printf
    popl      %eax
    popl      %edx
    pushl     $3
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

合，必要なデータを利用可能になるまで待つことによる実行の遅延を防ぐために，命令の順番の変更を行うことができますが，デバッグが難しくなってしまいます．

このオプションをデバッグ中に指定しておけば，GDBなどのデバッグ・ツールで混乱することもなく，正常にデバッグできます．

● -fno-sched-spec，-fsched-spec-load，-fsched-spec-load-dangerous

このオプションも同様です．

● -fno-trapping-math

このオプションを指定すると浮動小数点式のトラップを生成しません．ただし，本当に生成しなくて良いのか，よく考えて使用してください．

最適化の例となるプログラムをリスト15に示します．

● 生成するコンパイル・オプション（リスト16）

```
$ gcc test231.c -ftrapping-math -S
```

● 生成しないコンパイル・オプション（リスト17）

```
$ gcc test231.c -ffast-math -fomit-frame
                                  -pointer -O3
```

● -fno-zero-initialized-in-bss

このオプションは本来はBSSセクションに置かれた定数をゼロ・クリアしないためのものですが，無視されているようです．

BSSセクションに置いたものは，必ず初期化されるようです．初期化されたくない場合は，別の方法で行うべきです．

テスト・プログラムをリスト18に示します．

リスト15　浮動小数点式のトラップを生成しない最適化の例（test231.c）

```
//浮動小数点式のトラップを生成しない最適化をする例
#include <stdio.h>
#include <math.h>
const float f1 = 3.1212312312312312f;
const float f2 = 6.5432165432165432f;

float func(float a)
{
    float    x;
    x    =    f1 / f2;
     return a * f1 / f2;
}
main()
{
    float    z    =    func(3.1212312312312312f);
}
```

リスト16　トラップを生成したアセンブラ・ソース（test231.s）

```
    .file    "test231.c"
.globl f1
    .section .rodata
    .align 4
    .type    f1, @object
    .size    f1, 4
f1:
    .long    1078444609
.globl f2
    .align 4
    .type    f2, @object
    .size    f2, 4
f2:
    .long    1087463944
    .text
.globl func
    .type    func, @function
func:
    pushl    %ebp
    movl     %esp, %ebp
    subl     $4,   %esp
    flds     f1
    fdivs    f2
    fstps    -4(%ebp)
    flds     8(%ebp)
    fmuls    f1
    fdivs    f2
    leave
    ret
    .size    func, .-func
.globl main
    .type    main, @function
main:
    pushl    %ebp
    movl     %esp,        %ebp
    subl     $8,          %esp
    andl     $-16,        %esp
    movl     $0,          %eax
    subl     %eax,        %esp
    subl     $12,         %esp
    movl     $0x4047c241, %eax
    pushl    %eax
    call     func
    addl     $16,         %esp
    fstps    -4(%ebp)
    leave
    ret
    .size    main, .-main
    .section .note.GNU-stack,"",@progbits
    .ident   "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト17　トラップを生成しないアセンブラ・ソース（test231a.s）

```
    .file    "test231.c"
.globl f1
    .section .rodata
    .align 4
    .type    f1, @object
    .size    f1, 4
f1:
    .long    1078444609
.globl f2
    .align 4
    .type    f2, @object
    .size    f2, 4
f2:
    .long    1087463944
    .text
.globl func
    .type    func, @function
func:
    pushl    %ebp
    movl     %esp, %ebp
    subl     $4,   %esp
    flds     f1
    fdivs    f2
    fstps    -4(%ebp)
    flds     8(%ebp)
    fmuls    f1
    fdivs    f2
    leave
    ret
    .size    func, .-func
    .section .note.GNU-stack,"",@progbits
    .ident   "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

● 初期化するコンパイル・オプション(リスト19)

$ gcc test232.c -S -fno-zero-initialized-in-bss -fno-common

● 初期化しないコンパイル・オプション(リスト20)

$ gcc test232.c -S -fzero-initialized-in-bss -fno-common

また，オブジェクトのダンプをリスト21に示します．

このオプションを指定してもしなくてもアセンブラ・ソース中でゼロ・クリアされています．このオプションの意味がなくなったのか，バグなのかわかりませんが，あまり使わない処理なのかもしれません．

*　　　　*

次回も「最適化オプション」の続きを説明します．

きし・てつお

リスト18　BSSセクションに置かれた定数を初期化する例(test232.c)

```
//BSSセクションに置かれた定数を初期化する例
#include <stdio.h>
int a;
int b;
int c;
main()
{
    printf("%d¥n",a);
    printf("%d¥n",b);
    printf("%d¥n",c);
}
```

リスト19　初期化するアセンブラ・ソース(test232.s)

```
    .file     "test232.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8,   %esp
    andl      $-16, %esp
    movl      $0,   %eax
    subl      %eax, %esp
    subl      $8,   %esp
    pushl     a
    pushl     $.LC0
    call      printf
    addl      $16,  %esp
    subl      $8,   %esp
    pushl     b
    pushl     $.LC0
    call      printf
    addl      $16,  %esp
    subl      $8,   %esp
    pushl     c
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leave
    ret
    .size     main, .-main
.globl a
    .bss
    .align 4
    .type     a, @object
    .size     a, 4
a:
    .zero     4
.globl b
    .align 4
    .type     b, @object
    .size     b, 4
b:
    .zero     4
.globl c
    .align 4
    .type     c, @object
    .size     c, 4
c:
    .zero     4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト20　初期化しないアセンブラ・ソース(test232a.s)

```
    .file     "test232.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8,   %esp
    andl      $-16, %esp
    movl      $0,   %eax
    subl      %eax, %esp
    subl      $8,   %esp
    pushl     a
    pushl     $.LC0
    call      printf
    addl      $16,  %esp
    subl      $8,   %esp
    pushl     b
    pushl     $.LC0
    call      printf
    addl      $16,  %esp
    subl      $8,   %esp
    pushl     c
    pushl     $.LC0
    call printf
    addl $16, %esp
    leave
    ret
    .size     main, .-main
.globl a
    .bss
    .align 4
    .type     a, @object
    .size     a, 4
a:
    .zero     4
.globl b
    .align 4
    .type     b, @object
    .size     b, 4
b:
    .zero     4
.globl c
    .align 4
    .type     c, @object
    .size     c, 4
c:
    .zero     4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 21　生成されたオブジェクト・ダンプ・リスト 3(test232.txt)

```
test232:     ファイル形式 elf32-i386

セクション .init の逆アセンブル:

08048278 <_init>:
 8048278:    55                      push   %ebp
 8048279:    89 e5                   mov    %esp,%ebp
 804827b:    83 ec 08                sub    $0x8,%esp
 804827e:    e8 61 00 00 00          call   80482e4 <call_gmon_start>
 8048283:    e8 bc 00 00 00          call   8048344 <frame_dummy>
 8048288:    e8 c3 01 00 00          call   8048450 <__do_global_ctors_aux>
 804828d:    c9                      leave
 804828e:    c3                      ret
セクション .plt の逆アセンブル:

08048290 <.plt>:
 8048290:    ff 35 84 95 04 08       pushl  0x8049584
 8048296:    ff 25 88 95 04 08       jmp    *0x8049588
 804829c:    00 00                   add    %al,(%eax)
 804829e:    00 00                   add    %al,(%eax)
 80482a0:    ff 25 8c 95 04 08       jmp    *0x804958c
 80482a6:    68 00 00 00 00          push   $0x0
 80482ab:    e9 e0 ff ff ff          jmp    8048290 <_init+0x18>
 80482b0:    ff 25 90 95 04 08       jmp    *0x8049590
 80482b6:    68 08 00 00 00          push   $0x8
 80482bb:    e9 d0 ff ff ff          jmp    8048290 <_init+0x18>
セクション .text の逆アセンブル:

080482c0 <_start>:
 80482c0:    31 ed                   xor    %ebp,%ebp
 80482c2:    5e                      pop    %esi
 80482c3:    89 e1                   mov    %esp,%ecx
 80482c5:    83 e4 f0                and    $0xfffffff0,%esp
 80482c8:    50                      push   %eax
 80482c9:    54                      push   %esp
 80482ca:    52                      push   %edx
 80482cb:    68 0c 84 04 08          push   $0x804840c
 80482d0:    68 c4 83 04 08          push   $0x80483c4
 80482d5:    51                      push   %ecx
 80482d6:    56                      push   %esi
 80482d7:    68 70 83 04 08          push   $0x8048370
 80482dc:    e8 bf ff ff ff          call   80482a0 <_init+0x28>
 80482e1:    f4                      hlt
 80482e2:    90                      nop
 80482e3:    90                      nop

080482e4 <call_gmon_start>:
 80482e4:    55                      push   %ebp
 80482e5:    89 e5                   mov    %esp,            %ebp
 80482e7:    53                      push   %ebx
 80482e8:    e8 00 00 00 00          call   80482ed <call_gmon_start+0x9>
 80482ed:    5b                      pop    %ebx
 80482ee:    81 c3 93 12 00 00       add    $0x1293,         %ebx
 80482f4:    50                      push   %eax
 80482f5:    8b 83 fc ff ff ff       mov    0xfffffffc(%ebx),%eax
 80482fb:    85 c0                   test   %eax,%eax
 80482fd:    74 02                   je     8048301 <call_gmon_start+0x1d>
 80482ff:    ff d0                   call   *%eax
 8048301:    8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 8048304:    c9                      leave
 8048305:    c3                      ret
 8048306:    90                      nop
 8048307:    90                      nop

08048308 <__do_global_dtors_aux>:
 8048308:    55                      push   %ebp
 8048309:    89 e5                   mov    %esp,     %ebp
 804830b:    83 ec 08                sub    $0x8,     %esp
 804830e:    80 3d a0 95 04 08 00    cmpb   $0x0,0x80495a0
 8048315:    75 29                   jne    8048340 <__do_global_dtors_aux+0x38>
 8048317:    a1 9c 95 04 08          mov    0x804959c,%eax
 804831c:    8b 10                   mov    (%eax),   %edx
 804831e:    85 d2                   test   %edx,     %edx
 8048320:    74 17                   je     8048339 <__do_global_dtors_aux+0x31>
 8048322:    89 f6                   mov    %esi,     %esi
 8048324:    83 c0 04                add    $0x4,     %eax
 8048327:    a3 9c 95 04 08          mov    %eax,0x804959c
 804832c:    ff d2                   call   *%edx
 804832e:    a1 9c 95 04 08          mov    0x804959c,%eax
 8048333:    8b 10                   mov    (%eax),   %edx
 8048335:    85 d2                   test   %edx,     %edx
```

リスト 21　生成されたオブジェクト・ダンプ・リスト 3（test232.txt）（つづき）

```
 8048337:     75 eb                   jne    8048324 <__do_global_dtors_aux+0x1c>
 8048339:     c6 05 a0 95 04 08 01    movb   $0x1,0x80495a0
 8048340:     c9                      leave
 8048341:     c3                      ret
 8048342:     89 f6                   mov    %esi,      %esi

08048344 <frame_dummy>:
 8048344:     55                      push   %ebp
 8048345:     89 e5                   mov    %esp,      %ebp
 8048347:     83 ec 08                sub    $0x8,      %esp
 804834a:     a1 b0 94 04 08          mov    0x80494b0, %eax
 804834f:     85 c0                   test   %eax,      %eax
 8048351:     74 19                   je     804836c <frame_dummy+0x28>
 8048353:     b8 00 00 00 00          mov    $0x0,      %eax
 8048358:     85 c0                   test   %eax,      %eax
 804835a:     74 10                   je     804836c <frame_dummy+0x28>
 804835c:     83 ec 0c                sub    $0xc,      %esp
 804835f:     68 b0 94 04 08          push   $0x80494b0
 8048364:     ff d0                   call   *%eax
 8048366:     83 c4 10                add    $0x10,     %esp
 8048369:     8d 76 00                lea    0x0(%esi), %esi
 804836c:     c9                      leave
 804836d:     c3                      ret
 804836e:     90                      nop
 804836f:     90                      nop

08048370 <main>:
 8048370:     55                      push   %ebp
 8048371:     89 e5                   mov    %esp,        %ebp
 8048373:     83 ec 08                sub    $0x8,        %esp
 8048376:     83 e4 f0                and    $0xfffffff0, %esp
 8048379:     b8 00 00 00 00          mov    $0x0,        %eax
 804837e:     29 c4                   sub    %eax,        %esp
 8048380:     83 ec 08                sub    $0x8,        %esp
 8048383:     ff 35 a4 95 04 08       pushl  0x80495a4
 8048389:     68 98 84 04 08          push   $0x8048498
 804838e:     e8 1d ff ff ff          call   80482b0 <_init+0x38>
 8048393:     83 c4 10                add    $0x10,       %esp
 8048396:     83 ec 08                sub    $0x8,        %esp
 8048399:     ff 35 a8 95 04 08       pushl  0x80495a8
 804839f:     68 98 84 04 08          push   $0x8048498
 80483a4:     e8 07 ff ff ff          call   80482b0 <_init+0x38>
 80483a9:     83 c4 10                add    $0x10,       %esp
 80483ac:     83 ec 08                sub    $0x8,        %esp
 80483af:     ff 35 ac 95 04 08       pushl  0x80495ac
 80483b5:     68 98 84 04 08          push   $0x8048498
 80483ba:     e8 f1 fe ff ff          call   80482b0 <_init+0x38>
 80483bf:     83 c4 10                add    $0x10,       %esp
 80483c2:     c9                      leave
 80483c3:     c3                      ret

080483c4 <__libc_csu_init>:
 80483c4:     55                      push   %ebp
 80483c5:     89 e5                   mov    %esp,            %ebp
 80483c7:     57                      push   %edi
 80483c8:     56                      push   %esi
 80483c9:     53                      push   %ebx
 80483ca:     83 ec 0c                sub    $0xc,            %esp
 80483cd:     e8 00 00 00 00          call   80483d2 <__libc_csu_init+0xe>
 80483d2:     5b                      pop    %ebx
 80483d3:     81 c3 ae 11 00 00       add    $0x11ae,         %ebx
 80483d9:     e8 9a fe ff ff          call   8048278 <_init>
 80483de:     8d 93 20 ff ff ff       lea    0xffffff20(%ebx),%edx
 80483e4:     8d 8b 20 ff ff ff       lea    0xffffff20(%ebx),%ecx
 80483ea:     29 ca                   sub    %ecx,            %edx
 80483ec:     31 f6                   xor    %esi,            %esi
 80483ee:     c1 fa 02                sar    $0x2,            %edx
 80483f1:     39 d6                   cmp    %edx,            %esi
 80483f3:     73 0f                   jae    8048404 <__libc_csu_init+0x40>
 80483f5:     89 d7                   mov    %edx,            %edi
 80483f7:     90                      nop
 80483f8:     ff 94 b3 20 ff ff ff    call   *0xffffff20(%ebx,%esi,4)
 80483ff:     46                      inc    %esi
 8048400:     39 fe                   cmp    %edi,            %esi
 8048402:     72 f4                   jb     80483f8 <__libc_csu_init+0x34>
 8048404:     83 c4 0c                add    $0xc,            %esp
 8048407:     5b                      pop    %ebx
 8048408:     5e                      pop    %esi
 8048409:     5f                      pop    %edi
 804840a:     c9                      leave
 804840b:     c3                      ret
```

リスト 21　生成されたオブジェクト・ダンプ・リスト 3（test232.txt）（つづき）

```
0804840c <__libc_csu_fini>:
 804840c:     55                      push   %ebp
 804840d:     89 e5                   mov    %esp,           %ebp
 804840f:     56                      push   %esi
 8048410:     53                      push   %ebx
 8048411:     e8 00 00 00 00          call   8048416 <__libc_csu_fini+0xa>
 8048416:     5b                      pop    %ebx
 8048417:     81 c3 6a 11 00 00       add    $0x116a,        %ebx
 804841d:     8d 8b 20 ff ff ff       lea    0xffffff20(%ebx),%ecx
 8048423:     8d 83 20 ff ff ff       lea    0xffffff20(%ebx),%eax
 8048429:     29 c1                   sub    %eax,           %ecx
 804842b:     c1 f9 02                sar    $0x2,           %ecx
 804842e:     85 c9                   test   %ecx,           %ecx
 8048430:     8d 71 ff                lea    0xffffffff(%ecx),%esi
 8048433:     75 0b                   jne    8048440 <__libc_csu_fini+0x34>
 8048435:     e8 3a 00 00 00          call   8048474 <_fini>
 804843a:     5b                      pop    %ebx
 804843b:     5e                      pop    %esi
 804843c:     c9                      leave
 804843d:     c3                      ret
 804843e:     89 f6                   mov    %esi,           %esi
 8048440:     ff 94 b3 20 ff ff ff    call   *0xffffff20(%ebx,%esi,4)
 8048447:     89 f2                   mov    %esi,           %edx
 8048449:     4e                      dec    %esi
 804844a:     85 d2                   test   %edx,           %edx
 804844c:     75 f2                   jne    8048440 <__libc_csu_fini+0x34>
 804844e:     eb e5                   jmp    8048435 <__libc_csu_fini+0x29>

08048450 <__do_global_ctors_aux>:
 8048450:     55                      push   %ebp
 8048451:     89 e5                   mov    %esp,        %ebp
 8048453:     53                      push   %ebx
 8048454:     52                      push   %edx
 8048455:     a1 a0 94 04 08          mov    0x80494a0,   %eax
 804845a:     83 f8 ff                cmp    $0xffffffff, %eax
 804845d:     bb a0 94 04 08          mov    $0x80494a0,  %ebx
 8048462:     74 0c                   je     8048470 <__do_global_ctors_aux+0x20>
 8048464:     83 eb 04                sub    $0x4,        %ebx
 8048467:     ff d0                   call   *%eax
 8048469:     8b 03                   mov    (%ebx),      %eax
 804846b:     83 f8 ff                cmp    $0xffffffff, %eax
 804846e:     75 f4                   jne    8048464 <__do_global_ctors_aux+0x14>
 8048470:     58                      pop    %eax
 8048471:     5b                      pop    %ebx
 8048472:     c9                      leave
 8048473:     c3                      ret
セクション .fini の逆アセンブル:

08048474 <_fini>:
 8048474:     55                      push   %ebp
 8048475:     89 e5                   mov    %esp,           %ebp
 8048477:     53                      push   %ebx
 8048478:     e8 00 00 00 00          call   804847d <_fini+0x9>
 804847d:     5b                      pop    %ebx
 804847e:     81 c3 03 11 00 00       add    $0x1103,        %ebx
 8048484:     52                      push   %edx
 8048485:     e8 7e fe ff ff          call   8048308 <__do_global_dtors_aux>
 804848a:     8b 5d fc                mov    0xfffffffc(%ebp),%ebx
 804848d:     c9                      leave
 804848e:     c3                      ret
```

シニアエンジニアの技術草子 四拾五之段

◆賢い買い物

旭 征佑

●手軽なデジカメを購入

近所にTVでもよく宣伝している大型電気店があって，よく行っている．さすが売り上げが全国有数規模のチェーン店だけあって，フロアが広く品数も豊富だ．眩しいほど明るい店内では，どの店員の顔も景気がよさそうだ．買い物をすると，ポイントがつくのもうれしい．買う前についつい商品のポイントをチェックしてしまうし，レジでポイント・カードを返してもらうときは，すました顔をしながら実はちょっぴりわくわくしたりしているのだ．また，小物はポイントで買えたりするので，DVD-Rなど小物をタダで手に入れることもできる．勢い，店に行く回数が増えることになる．

そんなわけで，今日もまたこの店に来てしまった．今日は，急にデジカメが必要になったので，2万円未満の手ごろなものを探しにきたのだ．高価なものは技術革新が速いので，買うのはむだだ．いつでもどこでも撮影できるから少々不恰好でも乾電池が使えるものがいい．この二つは，デジカメ購入の場合の条件であり筆者の信念でもある(はずだった)．

ところが，この店では多くの機種が並んでいて，どれにしようか迷うことになった．そんなとき，若い店員がそろりと近づいてきた．彼は「絶対にこれがいいですよ」と，薄型の機種をすすめる．2万5千円ほどで，予算を超えるのだが，最新型で性能もいいと盛んにすすめてくるのだ．ケースがメタルだとか，スピードが速いとかいろいろ言う．でもこの機種は，乾電池が使えない．「いちいち充電しないと使えないのでは不便だ」というと，今度は店員は，「室内ではACアダプタをつないで使えます」といってくる．「それなら，まあいいかもしれない…」．悔しいが，確固たる信念が揺らいできているではないか．ところが，そのカメラの周りぐるっと見回して見ても，ACアダプタの接続コネクタらしき穴が見つからない．指摘すると，その店員は，「絶対大丈夫です．まちがいありません」．そう言うと，笑って相手にしてくれない．一方，ほかの製品に目移りしていると，「この製品は，ここがいいのですが，こういう大きな欠点がありまして…」と，追い込んでくる．結局，店員の圧倒的な攻めに根負けして，その機種を買ってしまった．あとで考えてみると，それが広告の商品で，ポイントがちょっと高いことも，購入動機の最後の一押しになったかもしれない．

さて，自宅でマニュアルを読んだら，専用のACアダプタを購入しないといけことがわかった．しかも，携帯電話の電池のような形をした特殊なブラケットをはめ込む形式だ，アダプタごとき分際で，なんと4,200円もするではないか．そうは言っても仕方がないので，買いにいった．店員には文句たらたら，たれてやろうなどという，浅はかな思いは，先ほどの店員が見つからないために空振りに終わる．仕方なしにほかの店員に聞くと，「在庫はありません．取り寄せに1週間以上かかりますが，取り寄せますか」という．そんな聞き方をされたら断るしかないではないか，などと思いながらも，素直に断って帰ってきた．実に間抜けな買い物をしたものだ．

恥ずかしい話しだが，こんな経験は，実は何回もしている．以前は，決してこんなことはなかったはずだ．電気製品を買うときは，多くのカタログを集めてじっくり読み込み，雑誌などで各種製品を研究した．友人から相談されても，細かいことまで議論し，自分なりの視点で薀蓄(うんちく)たっぷりお勧め商品を説明する．筆者はそうだったし，そんな友人も数多くいた．ところが最近はそんなわけにはいかなくなった．一体何が原因なのか．

●四つの理由

原因の一つは，製品の種類が多くなり，製品のサイクルが早くなったことが挙げられる．次々と新しい製品が出てくる．一昔前だったら，「名機」といわれる横綱的な製品がどの分野にも必ずあって，これに対抗する「新技術」を搭載したとする新製品が大々的な宣伝戦略にのってデーンと登場したものだ．名機は，憧れの的であり，何年間も安定して売れ続けた．製品を買うときは必ずこの名機が基準の一つになった．一方で，風呂敷よろしく理論的根拠を広げまくった新技術を使用した製品も，対抗馬に登場した(そういえば，新技術を「搭載した」という表現もなんだかすごそうだ)．こうやって，専門雑誌の裏表紙には，いつも見慣れた同じ製品の広告が載っていたものだ．しかし最近は，名機という言葉は死語になりつつあるし，毎度繰り出される新技術を搭載した製品の魔力も大きく減ってしまった気がする．製品寿命が短いということを如実に語っているのだろう．

次に挙げられる原因は，店員の質の問題だろうか．昔の専門店にいた，愛想は悪いが正確な知識をもった店員というのは，最近はほとんど見かけない．以前は秋葉原などに行くと，店員



が詳しい製品知識をもっていて，何を聞いても明快な回答が返ってきて頼もしかったものだ．店員自身が，店頭で技術書を読んでいたり，OSをインストールしていたりで自分の知識を広げることに対して興味をもっていた気がする．店員がある種の技術者だったのかもしれない．そんな店員に，さらに深く聞くと嫌味な顔をして答えてくれたりする．そしてさらに深く聞いたりすると，「買わねぇなら帰れ！」などと怒鳴られたこともあった．そんな店員に対抗するために，顧客の自分も知識をつけたものである．考えてみれば，この分野の市場も巨大化したわけで，そう店員の質を維持できるわけがないのも事実かもしれない．

3番目に挙げられる原因は，大型店の進出かもしれない．店員は当然ながら技術者より，販売員であることを求められる．ローテーションも多く，売り場を変わることも多いから，専門的な知識よりも，幅広い知識が要求されてきている．メーカの派遣店員にいたっては，自社の製品の紹介しかしないし，2度とその売り場に来ないことも多い．そんな店員が，客には見分けも付かずに店内にうろちょろしている状態が一般化している．こんな店員に捕まると，困ったことになってしまう．

そして最後に，もっとも悪いのは自分なのだ．忙しさのせいにして常日頃の興味のある製品のウォッチを怠るようになった．いつでもWebで製品の比較ができる．そんな気楽さがこんなことを許しているかもしれない．各種の雑誌が出ているが，製品サイクルと同じように，雑誌の賞味期限も短くなってきているのだから，雑誌もこまめにチェックしないといけない．大事なのは顧客である自分が，やたらと多品種の比較を行っていて無理があったり，その製品の良さを出すことのできないような比較実験などに惑わされないように知識を身につけるべきだ．また，店頭に行ったら，よくわかっている店員かどうかを見分けることも大事だ．人に親切にされると，ころっと参ってしまうという，日本人古来の性格が災いしないようにするために必要なことは，知識に裏付けられた信念ではないだろうか．ようするに怠慢にしているから悪いのだ．

●後悔先に立たず

ポイント・カードを大々的にはじめたのはヨドバシカメラが最初だと思う．それから業界には急速に普及した感がある．本来は安くて性能のいいものを探すべきなのだが，いつの間にか



ポイントを集めている自分に気が付いたりする．家電業界の急成長にはこのカードも一役買っていたのに違いない．

ポイント・カードが悪いとは思わない．いや，絶対なくなって欲しくない，とても良いしくみだと思う．問題は，大型店の進出で，サービスが置き去りにされているような気がする点だ．また，顧客自身の製品研究の努力が足りないような気がする．

よく売っている店員，いわゆるカリスマ店員というのが，必ずいるようだが，やはり徹底して商品知識を詰め込んでいる店員だという．そんな店員に捕まれば，こちらも半分成功なのかもしれない．しかしそうでない場合は，実に悲しい．ポイント・カードの誘惑に負けて，大型電気店へと向かい，ろくな商品知識もない店員のカモになって商品を買わされている無知な自分は，まったく「情けない」の一言に尽きる．これは何とか脱却しなければならない．そして賢い買い物をしよう．今日もまたこの店に来てそう誓うのだ．もし製品を買ったあとで後悔することがあるようなら，読者も要注意かもしれない．

あさひ・しょうすけ　テクニカル・ライタ
イラスト　森 祐子

NEWPRODUCTS

●組み込み用プロセッサ

Geode NX 1250 6W プロセッサ

- シンクライアント，プリンタ，POS，情報/取引KIOSK，通信・ネットワーク機器など，長期にわたるサポートが必要とされるソリューションへの搭載を念頭に設計された，x86アーキテクチャ対応のプロセッサ．
- AMD Athlonコアをベースとしており，低消費電力の組み込み用途から高性能サーバ・プラットホームに対応可能．
- 動作周波数1GHz以下の性能を追求する製品にとって，ファンの実装を不要とし，価格性能比に優れている．
- ファンなしで動作するため，設計コストの節減および高度な電力管理が可能．
- 消費電力は，標準で6W，TDPで9W．

●サンプル価格：￥5,355（10,000個時）

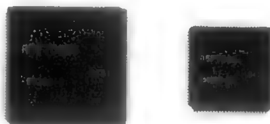
■日本AMD（株）
TEL：03-3346-7550
URL：http://www.amd.com/jp-ja/

●32ビット・マイコン

SH7080シリーズ

- 32ビットRISCコア「SH-2」を搭載し，最大周波数80MHzで動作する1チップ・マイコン．
- 常時1サイクル（12.5ns）でアクセス可能な，最大512Kバイトの高速フラッシュ・メモリを内蔵．
- 電源降圧回路を内蔵しており，3.3Vまたは5Vの単一電源で使用可能．
- 外付け電源回路やレベル・シフタなどの周辺ICの削減が可能．
- 外部にフラッシュ・メモリやSDRAMなどのメモリを直接接続可能なメモリ・コントローラを搭載．
- 汎用インバータやACサーボなどのモータ制御に最適化している．

●サンプル価格：￥1,733～￥2,415



■（株）ルネサス テクノロジ
TEL：03-5201-5214
E-mail：csc@renesas.com

●ディジタル・メディア・プロセッサ

TMS320DM642

- 動作周波数720MHzの，DSPベースのディジタル・メディア・プロセッサ．
- 高画質なストリーミング・ビデオの配信に適する．
- MPEG-4/2/1およびマイクロソフトのWMV HDによる解像度720pに対応し，H.264のビデオ・コーディングが可能．
- オン・チップのHD対応ビデオ・ポート，グルーレスEthernet接続，マルチチャネル・オーディオ，66MHz PCI対応など，各種マルチメディア向けペリフェラルや通信用ペリフェラルを統合．
- 高画質ストリーミング，ブロードキャスト・アプリケーションのパフォーマンスを高めるとともに，マルチチャネル・デコード，より高品質なオーディオ，ビデオ・アプリケーションの実現に必要な性能を提供．
- 開発ツールとして，ディジタル・メディア・デベロッパーズ・キットを提供．

●価格：$69.99（10,000個時）

■日本テキサス・インスツルメンツ（株）
URL：http://www.tij.co.jp/pic/

●シリアル・ディジタル・ビデオ・デコーダ

CLC031A

- HDTV機器保護機能である，6kV ESD耐量をもつシリアル・ディジタル・ビデオ・デコーダ．
- シリアル・ディジタル・ビデオ・エンコーダ「CLC030」と組み合わせることで，HDTV機器設計におけるデータ伝送インターフェースを大幅に簡素化．
- 270Mbps～1.485Gbpsの伝送速度でシリアル・データを受け入れ，SMPTE標準のデータ・レートでパラレル・データに変換．
- 広範囲な同相入力範囲を持つため，シリアル・インターフェース・シグナリングを幅広く選択できるほか，出力タイミング・マージンの拡大とジッタの低減によりビデオ・システムの処理性能が向上．
- 消費電力が低く，信頼性向上とシステム全体の部品コストの低減を実現．
- ブロードキャスト・ビデオ・オーナーズ・マニュアルを導入．

●価格：￥4,226（1,000個時）

■ナショナルセミコンダクタージャパン（株）
TEL：0120-666-116

●SD/SDIOホスト・コントローラ・デバイス

CG200

- SD/SDIOホスト・コントローラ・デバイス．
- SDバスは最大50MHzで動作可能で，最大20Mバイト/sの実行スループットを得ることが可能．
- SDメモリ，SDIOカードに対応する二つの独立したSDホストを搭載しており，高速SDメモリとSDIOなどのSDコンボ・アプリケーションを実現することが可能．
- PCI，IDE，組み込み機器向け汎用8/16/32ビット・バスなどのインターフェースを搭載しているため，デュアルSDスロットと組み合わせることによって，多様なシステムを実現．
- 二つのSDソケット，PCIコネクタ，IDEヘッダ，汎用8/16/32ビット・インターフェース・ヘッダを実装した，評価ボード「CG200EDK」を提供．

●価格：下記へ問い合わせ

■シイガイズ（株）
TEL：03-5575-3875
E-mail：sales@c-guys.com

●SMPTE物理層デバイス

HOTLink-On-Demandファミリ

- SMPTE標準に準拠したPHY．
- 独立4チャネル・ビデオ・シリアライザ/デシリアライザ「HOTLinkII」の実績をもとに構築され，プロダクション・スイッチャ，分配増幅器，D-A/A-Dコンバータ，カメラ制御器などのビデオ機器の設計にあたり，拡張性と柔軟性を提供．
- 各チャネルが，SMPTE 259MかSMPTE 292Mのいずれかの伝送レートで動作するように設計されている．
- すべてのデバイスが，統合電圧制御発振器（VCO）と位相ロック・ループ（PLL）の機能を備える．
- チャネル間の漏話を防止し，SMPTEのジッタ仕様より改善された設計となっている．
- 一方のチャネルがトランスミッタとなり，他方のチャネルが独立のリロッキング・デシリアライザとなれるデバイスを含んでおり，アップ・ダウンのコンバータ・アプリケーションに威力を発揮．

●価格：$19～$82（1,000個時）

■日本サイプレス（株）
TEL：03-5371-1921 FAX：03-5371-1955

■価格は税込みです．

●USB 2.0 コントローラ

EZ-USB FX2LP

- 低消費電力を意識したプログラマブルUSB2.0コントローラ.
- USB-IFにより認定.
- 480Mbpsのデータ・レート，16Kバイトのオンチップ・メモリ，最大40本までのプログラマブルI/Oを内蔵し，設計の柔軟性を提供.
- USB用プロセス・テクノロジで開発されたアドバンテージを持ち，他社製品と比較して約50%の消費電力の削減を実現.

●サンプル価格：$4.55～(10,000個時)

■日本サイプレス(株)
TEL：03-5371-1921 FAX：03-5371-1955

●Bluetooth-USBアダプタ

BT-02UD2

- 携帯情報機器やデスクトップPCなどのUSBポートに接続することで，Bluetooth機能を追加し，無線環境を構築可能なBluetooth-USBアダプタ.
- Bluetooth ver.1.1に準拠しており，最大通信速度は非対称型通信時に約723.2kbps，対称型通信時に約439.9kbpsを実現.
- IrDA規格と異なり，機器間に遮蔽物があっても通信可能.
- 本体サイズ20×10.7×39.0mm，重量11gの小型スティック・タイプで携帯性に優れる.
- 電力はUSBバスから供給されるため，外部電源は不要.
- ジェネリック・アクセス，サービス・ディスカバリ・アプリケーション，シリアル・ポート，ダイヤルアップ・ネットワーク，PIMアイテム転送，ファイル転送，FAXなどのプロファイル・サービスをサポート.

●サンプル価格：¥4,179

■プラネックスコミュニケーションズ(株)
TEL：0120-415976

●HDTV対応STB向けデコーダ

STi7710

- HDTV対応のSTB向けデコーダ用LSI.
- HDTVデコーダ用IC STi7020とOmegaデコーダSTi5517の機能を組み合わせ，Hi-Speed USBやHDCPプロセッサなどの機能を追加して1チップ化.
- HDCPに準拠した著作権保護機能を搭載.
- DVIとHDMIのディスプレイ・インターフェースを提供.
- HDMIはDVIに準拠して開発されており，最大8チャネルの192kHzオーディオを送信する機能をもつ.

●サンプル価格：$18



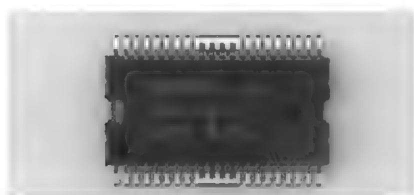
■STマイクロエレクトロニクス(株)
TEL：03-5783-8220 FAX：03-5783-8229

●ディジタル・アンプIC

YDA138

- フラット・パネルTV，液晶PCモニタ，ポータブル・オーディオ機器，PCスピーカなどの高音質用途向け，出力10W×2のディジタル・アンプIC.
- アナログ信号入力回路，パルス幅変調回路，ピュアパルス・ダイレクト・スピーカ・ドライブ回路，自励/他励クロック発振回路，ポップ・ノイズ低減回路，マスタ/スレーブ動作回路，キャリア周波数ホッピング回路，過電流保護回路，ヘッドホン・アンプなどの機能を集積.
- ピュアパルス・ダイレクト・スピーカ・ドライブ回路により，出力信号のパルス変調信号を制御し，*LC*フィルタを省略することで音質低下を防止.

●サンプル価格：¥630



■ヤマハ(株)
TEL：0539-62-5444

●電力量計向けIC

71M6513H

- 21ビットA-Dコンバータ，32ビット・コンピュート・エンジン，マイコン，RTC，LCDドライバおよび高精度基準電圧回路を集積した，産業用電力量計向けIC.
- 必要最小限の外付け部品で，0.1%未満の測定精度を実現.
- シングル・コンバータ技術とディジタル温度補正技術により，10ppm/℃の精度をもっている.
- 有効電力，電圧実効値，電流実効値といった標準測定値に加え，IC外部で処理を行うためのデータのカスタム出力が可能.
- 消費電流は動作時で30mW未満，バッテリ・モードで13μW.
- 64Kバイト・フラッシュ・メモリ，7KバイトRAM，RTC，LCDインターフェース，UART，I²Cインターフェースを内蔵.

●価格：$4.95





■TDK(株)
TEL：03-5202-7231

●マルチ・プラットホームFPGA

Virtex-4ファミリ

- 90nm/300mmプロセスによって製造された，マルチ・プラットホームFPGA.
- LX(ロジック主体アプリケーション用)，SX(高性能信号処理アプリケーション用)，FX(高速シリアル接続および組み込みプロセッシング向けアプリケーション用)の，三つのドメインに最適化されたプラットホームFPGAアーキテクチャを用意.
- 各FPGAは，500MHz DCM，PMCD，オンチップ差動クロック・ネットワーク，FIFOコントロール・ロジックを内蔵した500MHz SmartRAMテクノロジ，1Gbps I/Oなどの共通の機能がある.
- SXでは，SmartRAMメモリ・ブロックの増設により，最大512XtremeDSPスライスまでサポート可能.
- FXは，32ビットRISC PowerPCを最大二つ内蔵しており，1300Dhrystone MIPSを超える処理能力を実現するとともに，組み込み型10/100/1000 Ethernet MACコアを最大四つ搭載可能.

●価格：LX25 $39.99～(25,000個時)
SX25 $59.99～(25,000個時)

■ザイリンクス(株)
TEL：03-5321-7740 FAX：03-5321-7762

■弊誌では新製品に関するニュースリリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp (編集部)

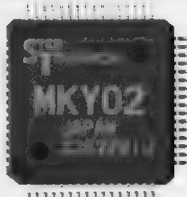
NEWPRODUCTS

●高速通信システムHUB用IC

MKY02

- 2,000点以上の信号すべてを1msで収集/配信可能な高速通信システム(Hi-speed Link System)対応のHUB向けのIC.
- T分岐を含む分配配線，スター配線，中継による延長が可能となり，1km以上の通信ケーブルを必要とする環境にも対応.
- 8個のポートを搭載.
- 信号中継時に劣化した信号を，完全補正する機能を備えている.
- 3.3V系と5V系TTL信号接続の両方が可能.
- 0.5mmピッチ，64ピンTQFP，外形寸法は12×12mm.

●価格：下記へ問い合わせ



■(株)ステップテクニカ
TEL：04-2964-8804 FAX：04-2964-7653
URL：http://www.steptechnica.com/

●PWM制御用IC

IR3092

- AMD Athlon，AMD Athlon64，AMD Opteron，Intel VR10.xプロセッサ向けの2相PWM制御用IC.
- 負荷電流が大きく，高効率が要求される省スペース型システムに適する.
- 発振器，PWM変調器2個，ハイサイド/ローサイドのゲート駆動回路2個，D-Aコンバータ，誤差アンプ，過電流検出用アンプなどを集積.
- 12V単一電源で動作し，ゲート駆動回路に電源を供給するリニア・レギュレータを内蔵.
- 発振周波数は，100kHz～540kHzの広範囲で設定可能.
- システム保護と信頼性向上のため，プログラム可能なソフト・スタート機能や過電流保護機能を装備.
- 電流検出は，高精度が得られるコイルを利用する.
- 低電圧ロックアウト機能，過電圧保護機能，パワー・グッド信号出力などを装備.

●サンプル価格：¥680

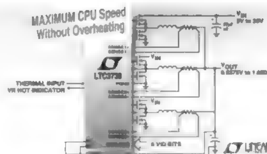
■インターナショナル・レクティファイアー・ジャパン(株)
TEL：03-3983-0086 FAX：03-3983-0642

●電源コントローラIC

LTC3738

- 高精度の電流バランシングとリモート・サーマル・モニタリングを備え，最大プロセッサ速度を達成するのに適する電源熱性能を保証するインテルVRM10 Pentium4電源コントローラIC.
- 60～200AのCPU負荷電流要件を満たすために，3～12フェーズに調整可能なPolyPhaseマルチ・チャネル降圧DC-DC同期整流式MOSFETコントローラ.
- 電力を各フェーズに均等に配分し，熱センサを使用できるため，システム部品の熱制限を超えずにCPUを最大速度で動作させることが可能.
- 最大4個の同時動作が可能なため，最適な熱性能で最大200Aの負荷電流を供給できる12フェーズ電源に使用することが可能.

●サンプル価格：¥537(1,000個時)



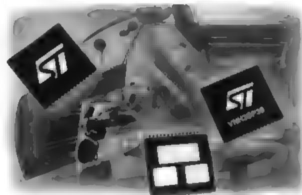
■リニアテクノロジー(株)
TEL：03-5226-7291 FAX：03-5226-0268

●車載用Hブリッジ

VNH3SP30

- ウィンドウ・リフト，シート・ポジショナの駆動，およびDCモータ制御などのハイパワー車載アプリケーション向けのHブリッジ.
- 出力電流30A，最大動作電圧40Vで，低損失動作時に，ハーフ・ブリッジごとに最大45mΩのオン抵抗を持つ.
- 制御入力は5Vロジック・レベルと互換性があり，最大10kHzでのPWM動作をサポート.
- アンダ/オーバ電圧，およびGNDまたはV_{cc}電位の損失に対し，広範囲な保護機能を備える.
- 過電圧クランプ，過熱保護回路，クロス・コンダクション保護，リニア電流リミッタを装備.

●サンプル価格：¥525(50個時)



■STマイクロエレクトロニクス(株)
TEL：03-5783-8240 FAX：03-5783-8216

●USBフラッシュ・メモリ・ディスク

BioFlash

- 台湾のグランド・ウェル・パワー・テクノロジー社が開発した，指紋認証センサ機能搭載のUSBフラッシュ・メモリ・ディスク.
- 容量は，128/256/512Mバイト，および1Gバイトの4種類を用意.
- 一度指紋を登録すれば，次回利用時からは，パソコンのUSBポートへのプラグ＆プレイで操作が可能.
- 付属のユーティリティ・ソフトにより，ディスク内のデータ・ファイルをドラッグ＆クリックの簡単操作で暗号化，圧縮/解凍が可能.
- パソコンのスクリーン・セーバをロックし，登録した指紋認証がない限り解除できないようにすることが可能.
- Address Book，eMail，Photo ViewerおよびSynchronizerの四つの機能を備えた付属ソフトを提供.

●価格：オープン価格



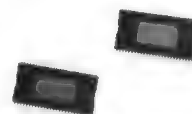
■(株)シスラボ
TEL：03-5985-6661 FAX：03-5985-6304

●アナログ入力型ディジタル・アンプIC

M61574FP M61575FP

- 薄型TVやアクティブ・スピーカなど向けの，アナログ入力型ディジタル・アンプIC.
- 「M61574FP」は最大10W×2チャネル出力，「M61575FP」は3チャネル・システムに特化した最大10W×2チャネル＋サブウーハ用20W出力.
- シングル・エンド方式の出力形式に単電源を採用するとともに，独自の高音質アーキテクチャ採用により出力雑音電圧50μVrms，全高調波歪率0.02%の高音質,高性能を実現.
- アナログ入力アンプからディジタル・アンプ・プロセッサ部とシングル・エンド2チャネルまたは3チャネル分のNチャネルMOSFETを含むパワー信号処理段を，1チップ化.

●サンプル価格：¥525(M61574FP)
¥735(M61575FP)



■(株)ルネサスソリューションズ
TEL：03-5201-5391
E-mail：csc@renesas.com

■価格は税込みです.

●GPSモジュール

S4E19863

- GPS付き携帯電話向けの，最高受信感度－160dBmのGPSモジュール．
- GPS測位が困難であった屋内やビル街でも，位置取得が可能．
- 3GPPに準拠した三つの測位モード（MS Based/MS Assisted/Autonomous）をサポートしているため，どのようなアプリケーションやネットワーク環境下でも，高水準のGPS測位パフォーマンスを発揮．
- 高速衛星サーチ・アルゴリズムを搭載．
- 測位時間（TTFF）は，屋外環境で2～3秒，屋内環境で7秒．
- 位置精度は，屋外環境で10m未満，屋内環境で35m未満．
- インターフェース電圧は1.8または3.0V，内部動作電圧は1.5または1.8V．
- モジュールは，GPSベース・バンド，GPS RF，およびTCXOを含まない周辺回路で構成．

●価格：下記へ問い合わせ

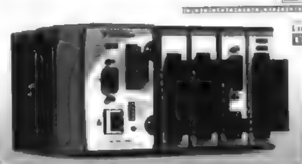
■セイコーエプソン（株）
TEL：042-587-1286
UEL：http://www.epsondevice.com/

●制御/集録プラットホーム

CompactRIO

- 小型で高性能なモジュール式の制御，集録プラットホーム．
- RIOテクノロジを採用し，VHDLなどの特殊なプログラミング言語の知識なしに，LabVIEWのグラフィカル開発環境を用いてプログラムしたロジックをFPGAにダウンロードでき，カスタム仕様の計測ハードウェア回路の定義が可能．
- 工業用の200MHz浮動小数点プロセッサを備えたリアルタイム・コントローラ「cRIO-9002」や「cRIO-9004」，4または8スロットの再構成可能シャーシ「cRIO-910xファミリ」が含まれる．

●価格：￥96,000～（シャーシ）
￥207,000～（コントローラ）
￥14,000～（I/Oモジュール）



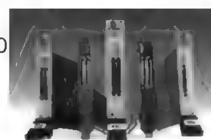
■日本ナショナルインスツルメンツ（株）
TEL：0120-527196 FAX：03-5472-2977
E-mail：salesjapan@ni.com

●データ集録モジュール

Mシリーズマルチファンクション DAQ

- 次世代マルチ・ファンクション・データ集録製品．
- 「NI-STC 2 ASIC」，「NI-MCalテクノロジ」，「NI-PGIA 2アンプテクノロジ」の三つの技術を採用し，入出力1チャネルあたり30%以上のコスト低減を実現．
- 最大32個の18ビット・アナログ入力チャネル，4個の16ビット・アナログ出力，2個の32ビットのカウンタ/タイマ，48のディジタルI/Oラインを装備．
- 最大32のディジタル・ラインのハードウェアによるタイミング制御を，最高10MHzで実行することが可能．
- NI-DAQmx計測ドライバ・ソフトウェアは，LabVIEW，Visual Studio .NET，NI Lab Windows/CVI対応のプログラミング・インターフェースとして使用可能．

●価格：
￥52,000～￥262,000



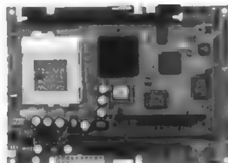
■日本ナショナルインスツルメンツ（株）
TEL：0120-527196 FAX：03-5472-2977
E-mail：salesjapan@ni.com

●組み込み向けCPUボード

SLC-8150-LVA

- Intel Pentium III（500MHz～1.26GHz），Celeron（300MHz～850MHz）CPUを搭載可能な，5インチ・ベイサイズのシングル・ボード・コンピュータ．
- PC/AT互換向けボードで，チップ・セットにVGAコントローラを内蔵するIntel 815E/ICH2を採用．
- 5.25”のサイズに，最大512MバイトのSDRAMの実装が可能．
- シリアル×4（RS-232-C×3，RS-232-C/422/485×1），USB×4，100Base-TX LAN×1，パラレル×1，EIDE×2，Disk On Chipソケット，AC97オーディオなど，豊富なインターフェースを標準装備．
- PISA，PC/104，MiniPCIの各拡張バス・スロットの装備により，機能拡張が行える．

●価格：
￥59,850



■（株）コンテック
TEL：03-5628-9286 FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●RFID簡易業務キット

RFIDトライアルキット

- RFタグのテスト導入を可能とする，RFID簡易業務キット．
- RFIDシステム構築に際し，必要なハードウェアとソフトウェアをパッケージ化．
- アプリケーション開発や他機器の購入が不要なため，低コストでの導入が可能．
- 業務アプリケーションとして「資産管理業務タイプ」，「出退勤管理業務タイプ」の2種類を用意．
- 「資産管理業務タイプ」では，非接触で耐環境性に強いRFタグで，棚卸などの資産管理が行えるアプリケーションを提供．
- 「出退勤管理業務タイプ」では，出退勤データの管理や日次，月次処理を可能とする専用端末を必要としない，パソコン1台で動作可能なアプリケーションを提供．

●価格：￥945,000



■富士通（株）
TEL：03-6252-2649 E-mail：
retail@fujitsu.com

●ARM評価ボード

A-plat

- マルチメディアおよび通信に関わる幅広い機能が搭載されている，フリースケール社のアプリケーション・プロセッサi.MX1（ARM920T）を搭載した評価ボード．
- A-plat（i.MX1），BSP，Windows CE .NETのスタータキット・ソリューションの提供，およびエンジニアリング・サポートを実施．
- Windows CE .NETのサポートにより，ほかのRTOSでは実現が難しかった機能を強化することが可能．
- Windows CE .NET開発者に対して，サポート・パッケージに加えて，ドライバやミドルウェアなどの開発受託およびコンサルティング・サービスなどを用意．
- A-plat，BSP，Windows CE .NETの組み合わせにより，豊富なアプリケーション，マルチメディア，インターネット接続に対応した，ハイ・パフォーマンスな小型携帯型コンシューマ機器などのプロトタイプ開発期間，製造期間の短縮を実現．

●価格：下記へ問い合わせ

■横河デジタルコンピュータ（株）
TEL：042-333-6202 FAX：042-352-6107
E-mail：info-wep@ydc.co.jp

■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

NEWPRODUCTS

●IP 電話開発向けプラットホーム

ボイス・オーバ・ワイヤレス LAN プラットホーム

- 無線 LAN IP 電話の開発向けに，VoIP 製品「TNETV1600」および無線 LAN 製品「TNETW1230WLAN」を内蔵した統合プラットホーム．
- 無線 LAN IP 電話市場向けの製品を短期間で市場投入することが可能．
- 「TNETV1600」は，「OMAP16xx」アーキテクチャをベースとしているため，コスト削減，柔軟性，消費電力管理を提供し，携帯電話と同等の通話時間と待受時間を備えた無線 LAN IP 電話の開発を可能にする．
- 「TNETW1230WLAN」チップセットとソフトウェア・サポートを内蔵し，必要とされる品質の音声を無線 LAN ネットワークで実現する音声処理ソフトウェア「Telogy Software for VoIP」などに対応．

●価格：下記へ問い合わせ

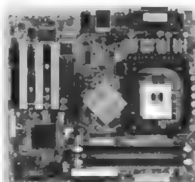
■日本テキサス・インスツルメンツ（株）
FAX：0120-81-0036
URL：http://www.tij.co.jp/pic/

●組み込み向けマザーボード

G4S306-C 型

- 台湾ディエフアイ（DFI）と共同開発した，組み込み向けのマザーボード．
- 3年間を基準とした，長期供給を保証．
- 独自の品質基準を適用しており，国内顧客の品質要求を実現．
- DFI の豊富なマザーボードのラインナップから，顧客固有のカスタマイズ要求に対応．
- 組み込み OS（Windows/Linux）のサポートや動作保証が可能．
- BIOS のカスタマイズ要求に対応可能．
- 特定仕様のボードや筐体などの開発対応が可能．
- 従来と比較して，約 20～30%のコスト低減を実現．

●価格：下記へ問い合わせ



■東芝パソコンシステム（株）
TEL：03-3279-9726

●ディジタル・ビデオ伝送システム

CamOnIP on PC-CUBE

- 光ファイバ回線に対応した，DV 伝送システム．
- DV カメラを利用して，双方向の高品位動画像伝送を行うことが可能．
- テレビ会議システム，遠隔講義システム，高精細医療画像伝送システムなど，高精細な動画を B フレッツ広域回線で可能としている．
- 電源を入れるだけで，送受信が可能．
- 本体装置は，約 10cm 角の手のひらサイズのキューブ型で，省スペースでハイ・パフォーマンスなシステムを提供．
- キーボードやマウスは不要．
- iLink ケーブルを接続するだけで，自動的にカメラを認識し通信を開始するなど，ホット・プラグおよびプラグ＆プレイに対応．

●価格：下記へ問い合わせ

■FA システムエンジニアリング（株）
TEL：089-931-2886 FAX：089-941-0336
E-mail：faseinfo@fase.co.jp

●ディジタル家電 統合プラットホーム

UniPhier

- 携帯電話から家庭用 AV 機器まで，幅広いディジタル家電に対応する，プロセッサ，OS，ミドルウェアなどを含めた統合プラットホーム．
- トータル開発効率を 5倍以上に向上．
- 商品分野ごとに異なっていた，ソフト・インターフェースとソフト構造を共通化し，ミドルウェア，OS，ドライバで構成されるソフト・プラットホームを確立．
- 分野別専用 DSP で培った AV 処理技術を集約した，UniPhier プロセッサを開発．
- UniPhier プロセッサは，C/C++ 言語に対応し，AV 用途に最適化した命令並列プロセッサをベースに，データ並列プロセッサ，ハード・エンジンなど拡張性に富んだスケーラブル・アーキテクチャで，分野に応じた展開が可能．

●価格：下記へ問い合わせ

■松下電器産業（株）
TEL：075-951-8151
E-mail：semiconpress@scd.mei.co.jp

●USB ホスト側プロトコル・スタック

PrUSB/Host

- プリンタや複合機などの高機能組み込み機器向けの，USB ホスト側プロトコル・スタック．
- USB デバイスの挿抜検出が可能．
- Control 転送での，USB 標準リクエストの管理が可能．
- ホスト-デバイス間のデータ・フローの管理．
- HUB をサポートし，複数の USB デバイス機器の管理が可能．
- UHCI コントローラに対応．
- Control，Interrupt，Bulk の 3 種類の転送タイプをサポート．
- Mass Storage クラス・ドライバのサンプルを標準添付．
- ホスト・コントローラや RTOS 依存モジュールを分割しているため，高い移植性を保持．

●価格：下記へ問い合わせ

■イーソル（株）
TEL：03-5302-1360 FAX：03-5302-1361
E-mail：ep-inq@esol.co.jp

●ブロードバンド・ルータ

CentreCOM AR260S

- 最大 98Mbps のスループットを実現した，高速ブロードバンド・ルータ．
- 10/100Base-TX を WAN 接続用に 1 ポート，LAN 接続用に 4ポート装備．
- WAN ポートは PPPoE 接続をサポートし，マルチ・セッション，アンナンバード，キープ・アライブに対応．
- 通信速度は，ENATとファイアウォールを使用時に 1518 バイトのパケット長で最大 98Mbps を実現．
- VPN 環境においても，認証（MD5，SHA-1），暗号化（3DES，DES）をハードウェア処理することで，最大 56Mbps のスループットを実現．
- アグレッシブ・モードのサポートにより，固定の IP アドレスが割り振られないインターネット接続サービスを利用して低コストの VPN の構築が可能．
- ステートフル・インスペクション，DoS アタック・プロテクト，パケット・フィルタリング，URL フィルタ，セルフ・アクセス制御，ステルス・モードを装備し，インターネットを経由した外部からの不正アクセスを遮断．

●価格：￥31,290

■アライドテレシス（株）
TEL：0120-860442

■価格は税込みです．

●組み込みLinux開発環境

Lineo uLinux ELITE Lite

- 組み込みLinux統合クロス開発環境で，Lineo uLinux ELITEのエントリ版．
- 本体と対応評価ボード用「BSP Lite」で構成されており，Linuxを製品に組み込むために必要な基本的な機能を装備．
- 対応カーネルは，Linux 2.4.22．
- プロジェクト生成ウィザード，コンフィグレーション・ツール，パッケージ管理ツール，ターゲット・イメージ・ビルダなどのシステム開発ウィザード，ツールを搭載．
- 対応ホストとして，Windows 2000/XP，Red Hat Linux 8.0/9.0，Fedora Core 1/2に対応．

●価格：¥105,000

■リネオソリューションズ(株)
TEL：03-5730-0123 FAX：03-5730-0125

●DSPソフトウェア開発ツール

Code Composer Studio Tuning Edition

- 組み込みシステム・アプリケーションの開発期間の短縮を実現する，ソフトウェアと開発ツールの標準統合開発ツール．
- 七つのツールからなる統合型ツール群の提供により，DSPソフトウェア開発プロセスの迅速化を実現．
- プログラム・コードに対して検証を行い，改善余地のある部分を特定してアドバイスを行い，システム・パフォーマンスやメモリの使用状況の最適化を支援．
- アドバイス・ウィンドウ，ゴール・ウィンドウ，プロファイル設定，プロファイル・ビューワの四つのウィンドウで構成される，設定可能なディスプレイ「ダッシュボード」を搭載．
- 「ダッシュボード」は，コンパイラ・コンサルタント，コードサイズ・チューン，キャッシュ・チューンと協調して機能する．

●価格：¥483,000

■日本テキサス・インスツルメンツ(株)
FAX：0120-81-0036
URL：http://www.tij.co.jp/pic/

●アプリケーション開発ソフトウェア

SDモバイルソリューション

- 携帯電話のSDメモリーカードのアプリケーション開発用ソリューション．
- 携帯電話向けでは，SD-Binding規格に初めて対応したミドルウェア．
- SD-Binder，SD-Audio，SD-Voice対応のミドルウェア，SDドライバ，セキュア・ライブラリの5種類のソフトウェア群で構成．
- ハードウェア制御用のドライバから，プログラミングが容易なAPIレベルのソフトウェア群により，アプリケーション開発のソリューションを提供．
- SD-Audio，SD-Voice対応ミドルウェアは，携帯電話で使用頻度の高い機能に対応したソフトウェアで，携帯電話でのSDメモリカードを使用したICレコーダやシリコン・オーディオなどの多彩な機能を短時間で実現可能．

●価格：下記へ問い合わせ

■(株)ルネサス テクノロジ
TEL：03-5201-5234
E-mail：csc@renesas.com

●ログオン管理ソフトウェア

UserLock v3.0

- フランスのISディシジョン社が開発した，Windowsユーザのログオン状況を管理するソフトウェア．
- Windows NT/2000/XP/Server 2003ネットワーク全体で，同一ユーザ名の同時接続数を制限．
- 指定されたユーザのログオン/ログオフを，メールで通知．
- MMCのスナップインにより，ユーザ・グループの特定が可能．
- コンピュータ名，IPアドレスでの範囲制限が可能．
- ターミナル・サーバ(Citrix ICA/Microsoft RDP)のセッション管理をサポート．
- 1台のUserLockサーバから，複数のドメインを管理可能．
- 管理コンソールからのユーザ・ログオフ機能をサポート．
- ログオン/ログオフのCSVログ・ファイルを出力．

●価格：¥9,450(10ユーザ)
¥39,900(50ユーザ)

■(株)エージーテック
TEL：03-3293-5283 FAX：03-3293-5270
E-mail：info@agtech.co.jp

●ジッタ解析ソフトウェア

TDSJIT3 v2

- オシロスコープ向けの，ジッタ解析ソフトウェア・パッケージ．
- 高速で正確なジッタ測定や，タイミング測定を実行することが可能．
- 測定ウィザードを使用することにより，クロック信号やデータ信号の複雑なジッタを短時間で解析し，設計，デバッグ，特性評価，設計検証を容易に実行可能．
- 従来，手作業に頼っていた各種機能を自動化．
- 測定とソースのコンフィグレーション・スクリーンは個別に用意されているため，どちらのコンフィグレーション設定が測定結果に影響を与えているのかの判断が容易．
- コンフィグレーション・スクリーンにより，設定に関する情報が得られるため，オシロスコープ設定やアプリケーション設定を最適に行うことが可能．
- 新機能として，スペクトル・アベレージングとスペクトル・ピーク・ホールドを追加．

●価格：¥1,081,500

■日本テクトロニクス(株)
TEL：03-6714-3010 FAX：0120-046-011

●Excel連携ツール

KeySQL Ver5.0

- Oracle，DB2，Microsoft SQL Serverなどのデータベースにアクセスし，データの検索，加工，更新が可能なExcelの連携ツール．
- SQLを意識することなく，ドラッグ&ドロップ主体のGUI操作で，データベースのデータをExcelのワークシートに取り込み，計算やレポートの作成を可能にする．
- Citrix MetaFrameやMicrosoft Terminal Serverに対応し，複数のOSのクライアントからの利用が可能．
- 検索条件をマクロファイルとして保存し，再利用が可能．
- VBAマクロ自動生成機能により，Excelアプリケーションの開発が可能．
- 検索結果やテーブル内容を参照可能なプレビュー機能，データ分析や集計に適するクロス集計機能などをサポート．

●価格：¥63,000

■(株)ネットワールド
URL：http://www.networld.co.jp/

■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

11/9-10 **RFID Developer Conference**
San Mateo Marriott, San Mateo, CA / San Francisco Airport, San Francisco, CA, USA
Shorecliff Communications
http://www.rfid-world.com/rfidepcdev/

11/9-10 **Wireless Connectivity Americas**
Santa Clara Convention Center, Santa Clara, CA, USA
IBC Telecoms & Media Group
http://www.wiconamericas.com/

11/11-14 **2004 China Robot Expo**
全国農業展覧館，北京市朝陽区，中国
中国科学院自動化研究所，中国国際企業合作公司
http://www.robotexpo.jp/

11/14-17 **SEMI NanoForum 2004**
Hilton Austin Hotel, Austin, TX, USA
Semiconductor Equipment and Materials International
http://wps2a.semi.org/wps/portal/

11/17-19 **Flat Information Displays Conference 2004**
Hotel Nikko San Francisco, San Francisco, CA, USA
iSuppli Corporation
http://www.isuppli.com/fid/

11/30-12/2 **Wi-Fi Planet Conferences & Expo Fall 2004**
San Jose McEnery Convention Center, San Jose, CA, USA
Jupitermedia Corporation
http://www.jupiterevents.com/wifi/fall04/

12/8-10 **Digital Video Expo West 2004**
Los Angeles Convention Center, Los Angeles, CA, USA
CMP Media
http://www.dvexpo.com/west/

国内イベント

11/3-7 第38回東京モーターショー2004
一働くくるまと福祉車両一
幕張メッセ(千葉県千葉市美浜区)
(社)日本自動車工業会
http://www.tokyo-motorshow.com/

11/10-12 第15回マイクロマシン展
科学技術館(東京都千代田区北の丸公園)
メサゴ・メッセフランクフルト(株)
http://www.micromachine.jp/

11/11-13 2004 NEW環境展 福岡会場
マリンメッセ福岡(福岡県福岡市博多区)
日報イベント(株)
http://www.nippo.co.jp/nexpo004/

11/17-19 Embedded Technology 2004
パシフィコ横浜(神奈川県横浜市西区)
(社)日本システムハウス協会
http://www.jasa.or.jp/et/

11/29-3 Internet Week 2004
パシフィコ横浜(神奈川県横浜市西区)
(社)日本ネットワークインフォメーションセンター
http://internetweek.jp/

12/1-3 SEMICON Japan 2004
幕張メッセ(千葉県千葉市美浜区)
Semiconductor Equipment and Materials International
http://wps2a.semi.org/wps/portal/

12/7-9 TRONSHOW 2005
東京国際フォーラム(東京都千代田区丸の内)
(社)トロン協会/T-Engineフォーラム
http://www.tron.org/

開催日，イベント名，開催地，問い合わせ先の順

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セミナー情報

最新組み込み設計技術セミナー
開催日時　：11月2日(火)
開催場所　：パシフィコ横浜アネックスホール(神奈川県横浜市西区)
受講料　：無料
問い合わせ先：CQ出版社TSE事務局，☎(03)5395-1465，FAX(03)5395-3911
http://it.cqpub.co.jp/tse/EDT200411/

TCP/IPによるI/O制御の実際～Ethernetを利用した組み込み機器の設計
開催日時　：11月5日(金)～11月6日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：25,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255
http://it.cqpub.co.jp/eSeminar/

CCDイメージセンサの基礎と応用
開催日時　：11月18日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：12,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

CMOSイメージセンサの基礎と応用
開催日時　：11月19日(金)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：12,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

C言語ベースのシステムLSI設計
開催日時　：11月26日(金)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

ステートマシンの設計技術
開催日時　：11月27日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

無線ICタグの仕組みと動向
開催日時　：12月2日(木)
開催場所　：NEC住友芝公園ビル NECラーニングセンター(東京都港区)
受講料　：36,750円
問い合わせ先：NECEラーニング事業部，☎(03)5232-3072，FAX(03)5232-3082
http://www.sw.nec.co.jp/el/

電磁界シミュレータでわかる高周波技術
開催日時　：12月2日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：12,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

HDEセキュリティソリューションセミナー
開催日時　：12月9日(木)
開催場所　：(株)ホライズン・デジタル・エンタープライズ(東京都渋谷区神山町)
受講料　：無料
問い合わせ先：(株)ホライズン・デジタル・エンタープライズ，☎(03)5738-5410，FAX(03)5738-5412
http://www.hde.co.jp/seminar/

SH-Linuxマイコン入門
開催日時　：12月9日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

Linuxデバイスドライバ入門
開催日時　：12月10日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

USBの基礎と応用
開催日時　：12月16日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

情報セキュリティの基礎
開催日時　：12月17日(金)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

やり直しのための積分変換
開催日時　：12月18日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

C++Builder6アプリケーション開発
開催日時　：12月20日(月)～12月22日(水)
開催場所　：新宿ファーストウエスト(東京都新宿区西新宿)
受講料　：99,750円
問い合わせ先：ボーランド(株)トレーニングセンター，☎(03)4560-1122，FAX(03)4560-1124
http://psc.borland.co.jp/

読者の広場

Interface への声



2004年10月号特集「USBホスト&ターゲット・システム設計技法」に関して

▷USB On-The-Goが規格化されてしばらくたちますが，身近な製品として目に触れることはまだありません．非常に興味深い規格だけに，キラー・アプリケーションの出現による普及が望まれるところです．（moto）

▷コネクタ・ピン配置大全集は大変良いし役に立つ．こういった付録をシリーズ的に企画してほしい．今後が楽しみ．（てんてん）

アンケートの結果

興味のあった記事（2004年10月号で実施）

①プロローグ USB機器のハードウェアとソフトウェアの構成
②第2章 SuperH & H8Sマイコンを使ったUSB機器の開発事例
③第3章 SL811を使った簡易ホストとUSBキーボードの接続実験
④第1章 ISP1582を使ったUSB機器の開発事例
⑤第4章 ハイ・スピード対応ホスト・コントローラM66596の概要
⑥第5章 On-The-Goの概要とML60842を使ったOTGシステムの開発事例
⑦特別付録 コネクタ・ピン配置大全集
⑧フリーソフトウェア徹底活用講座（第19回）
⑨開発技術者のためのアセンブラ入門（第29回，最終回）
⑩Appendix1 USB CVとUSBアナライザを使ったデバッグ技法
⑪第6章 ISP1362の概要とOn-The-Goサンプル・プログラムの詳細

特集『USBホスト&ターゲット・システム設計技法』についてのアンケートの結果

Q1　USBターゲット機器を設計/開発されたことがありますか？
①はい（45%）　②いいえ（55%）
（Q1で「はい」と回答された方に質問です）
Q2　ターゲットのファームウェア開発はどうされましたか？
①自社開発（80%）
②市販スタックを購入（20%）
③ファームウェア不要のUSBコントローラを採用（0%）
Q3　USBホスト機器を設計/開発されたことがありますか？
①はい（18%）　②いいえ（82%）
（Q3で「はい」と回答された方に質問です）
Q4　ホストのドライバ/スタック開発はどうされましたか？
①自社開発（50%）
②市販スタックを購入（0%）
③OSに付属（WindowsCE，Linuxなど）（50%）

読者プレゼント




●応募方法：本誌読者アンケートはがきに必要事項を記入のうえ，2004年11月30日（必着）までにご応募ください．なお，当選者の発表は発送をもってかえさせていただきます．

（1）Virtex-4ロゴ入り64Mバイト USBメモリ　（1名）
（2）卓上電子時計　（1名）
ザイリンクス（株）
（http://www.xilinx.co.jp/）

特集担当デスクから

☆近年話題になっているRFIDですが，「すべての製品にRFIDタグを付与」，「膨大な数のタグを使ってデータを管理」と，大規模な市場を視野に入れているところが目に付きます．
☆もちろん，RFID技術は膨大なデータを管理することも可能な技術です．だからといって，そういった「膨大なデータ」を扱う人たちだけのものではありません．「ウチでは大規模な物流は行わないから無関係…」とは思わずに，まずは手軽に試してみようということから，USB接続のRFIDリーダ/ライタ，H8マイコンによる製作キットによる解説記事を掲載しました．前者はVisual Basic，後者はC言語での開発が可能です．価格も数万円で手に入り，気軽に試してみることができます．
☆また，応用事例として，「食品トレーサビリティ」や「病院・介護ホーム支援システム」などのシステムについてと，今月号Show&Newsのページで，特集第4章で解説したユビキタスIDを使った神戸でのプレ実証実験についての記事も取り上げています．あわせてご覧ください．

Interface

次号予告

2005年1月号は
11月25日発売です

組み込みソフトウェアのクロス開発技法

クロス開発/スタートアップ・ルーチン/ライブラリ/gcc/gdb/デバッグ技法/ROM化

GNU/Linuxをはじめとするオープン・ソース・ソフトウェアの普及により，Cコンパイラやデバッガなどのソフトウェア開発ツールが，オープン・ソースのものだけで揃うようになった．このような有用なソフトウェア群を活用しない手はない．

しかし，単にgccが動くというだけでは，実際のクロス開発にはなかなか使えない．PC上であれば，それぞれのOS上で動作する標準入出力やファイル・アクセスのためのライブラリが用意されている．しかし組み込み機器では，プラットホームごとにハードウェア仕様が異なるため，標準入出力ライブラリすらまともに用意されていないことが多い．

さらにデバッガとして代表的なgdbも，ターゲット・ボードとホスト環境をどのようなインターフェースで接続するかなどにより，gdbスタブ部分をどう移植するかが問題となる．

そこで次号では，クロス開発とは何かといった話題から，自社で設計したシステム・ボードに各種ライブラリなどを移植する方法，そしてgdbスタブの移植方法や，それを活用したデバッグ方法などについて詳しく解説する．

編集後記

●始皇帝は秦王として13歳で即位すると，ただちに驪山のふもとに自身のための地下宮殿のような陵墓の造営を始め，死ぬまで造営を続けさせた．兵馬俑が発見されてから，跪座俑，百劇俑，文官俑などさまざまな姿をした秦俑が相次いで出土した．「大兵馬俑展」は始皇帝の見た死後の世界を見るような思いにさせられた．（檀）

●増刊編集作業が遅れに遅れ，本誌と同時進行状態になっております（汗）．で，誌面もさることながら，増刊付属CD-ROM収録のサンプル・プログラムが記事のとおりの動作をしなくてあせりまくり…マスタアップの土壇場（ちょっとオーバしたという話も）でなんとかデバッグ完了．こんなんでホントに間に合うんか?（汗）（M）

●それにしても今年の気候の異様さにはまいってしまう．真夏日の日差しに焼かれた次の日には肌寒くなってしまい，体調に大きく影響が出てしまっている．今年の酷暑は，生きている心地がしなかったが，急に寒くなられてしまうと，身体が順応できずにそれだけで体調を崩してしまう．暖冬になることを願う．（＝I0）

●職業柄，最新の技術に対して「トレンドを追いかける」という程度には追従しているのですが，「自分が使うため」が目的でないため，「耳学問」になってしまっているのではないかと反省．いざとなったら自分で使えるくらいには習得しておきたいものです．腕が鈍らないように，リハビリも兼ねて何かやっておくかな．（み）

●朝起きて，梨を食べようと手を伸ばしたとたん，ぎっくり腰に．現在，湿布を貼り，腰痛ベルトを巻いて生活している．立っているのは平気なのに座ると痛い．スニーカーよりもヒールのついた不安定な靴のほうが歩きやすい（←なぜ？）．階段は上りよりも下りが辛い…下りエスカレータのない駅が憎いっっ!!（もみ）

●朝起きたらガスが止まってました．火が使えないから調理もできないし，シャワーも使えない．まさか料金滞納したのか！っとも考えたけど，通帳を見たらきちんと引かれてるし．困ったあげくガス会社に電話したところ，前日ガス漏れしたときに止まったとのこと．そういえば，前の晩に警報機が鳴ったんだっけ．（Y2）

●子供が2歳になった．この間，2人で回転寿司に行ったのだが，席に座るなり店員に「子供用のフォークとシュプーンくだしゃい」と発声している．いつの間にそんな言葉を！ だいたい「子供用」って意味わかってるのか！と心の中で突っ込みながら「父親はなくとも子は育つ」を噛みしめた微妙な瞬間であった．（ちゃん）

●友人のパソコンのデータがふっ飛んでしまいました．なんとかこの夏を乗り切ったと思ったら，気を抜いたとたんに故障してしまったようです．秋になってから夏の疲れが出てくるあたりが妙に人間くさい．飼い主，もとい持ち主に似たのだろうか．そのうち「食欲の秋だから」と，メモリを大量に消費するようになるのかな．（と）

お知らせ

■ 読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■ 投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■ 本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■ コピー・サービスのご案内

本誌バックナンバの掲載記事については，在庫（原則として24か月分）のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金（税込み）
  1ページにつき100円
- 発送手数料（判型に関わらず）
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461 東京都豊島区巣鴨1-14-2
  CQ出版株式会社 コピー・サービス係
  （TEL：03-5395-4211，FAX：03-5395-1642）

■ お問い合わせ先のご案内

- 在庫，バックナンバ，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株） 2004 振替 00100-7-10665
2004年12月号 第30巻 第12号（通巻第330号）
2004年12月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏 村上真紀 山口光樹 大竹友美 小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／藤波 淳 森 祐子
広告／澤辺 彰 中元正夫 菅原利江

発行所／CQ出版株式会社 〒170-8461 東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122 FAX／（03）5395-2127
広告部（03）5395-2133 URL http://www.cqpub.co.jp/interface/
販売部（03）5395-2141 E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社 美和印刷株式会社
製本／星野製本株式会社


日本ABC協会加盟誌
（新聞雑誌部数公査機構） ISSN0387-9569